VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ZABEZPEČENÍ PROSTORU POMOCÍ VIDEOKAMERY A OS LINUX

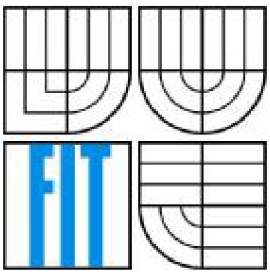
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN VALEŠ

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ZABEZPEČENÍ PROSTORU POMOCÍ VIDEOKAMERY A OS LINUX

VIDEOCAMERA BASED SECURITY GUARD FOR OS LINUX

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. JAN VALEŠ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JAROSLAV ŠKARVADA

BRNO 2007

Zadání diplomové práce/4848/2006/xvales00

Vysoké učení technické v Brně - Fakulta Informačních technologii

Ústav počítačových systémů Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Valeš Jan, Bc.**
Obor: Počítačové systémy a sítě
Téma: **Zabezpečení prostoru pomocí videokamery a OS Linux**
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s aplikačním programováním pro operační systém Linux.
2. Nastudujte rozhraní BSD sockets, PThreads, VIdeo4LInux2.
3. Navrhněte a Implementujte robustní metodu pro detekci pohybu v obrazu.
4. Zvolte vhodný formát pro přenos real-time video streamu přes síť.
5. Navrhněte vhodný systém zabezpečení video serveru, způsob autorizace klientů a zvažte i možnost použití šifrovaného přenosu.
6. Navrhněte architekturu Video serveru, který umožní snímat obraz z V4L2 kompatibilního zařízení (kamera), streamovat video přes síť na libovolný počet klientu ve vhodném formátu a také umožní detekovat pohyb v obrazu.
7. Navržený Video server implementujte.
8. Funkčnost systému vhodně demonstrujte.

Literatura:

- Dle pokynu vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění bodů 1-5.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.flt.vutbr.cz/Info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Škarvada Jaroslav, Ing.,** UPSY FIT VUT
Datum zadání: 28. února 2006
Datum odevzdání: 22. května 2007

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologii Vysokého učení technického v Brně.

Abstrakt

Práce se zabývá implementací aplikace, která umožňuje zabezpečení prostoru pomocí web kamery a OS Linux. Hlavní součástí aplikace je proces běžící v pozadí, který komunikuje s kamerou přes V4L rozhraní. Pomocí dynamicky nahrávaných modulů se provádí detekce pohybu ve snímané scéně. Algoritmus detekce lze jednoduše měnit pomocí uživatelských modulů. Aplikace umožňuje ukládat snímaná data ve formě obrázků nebo video souborů.
Pro monitorování snímané scény v reálném čase byl implementován klient, který umožňuje připojení přes TCP/IP k hlavní aplikaci. V komunikačním protokolu je implementována i jednoduchá autentizace klientů a šifrování přenášených dat.

Klíčová slova

Linux, GTKmm, detekce pohybu, snímání obrazu,V4L, , video streaming, autentizace.

Abstract

This thesis deals with the implementation of security guard software for OS Linux using an appropriate web camera. The main part of this application is process running in background using V4L application interface to communicate with web cam. Because this program uses dynamically loaded plug-ins for motion detection, it is very simple to change detection algorithm just by modifying configuration file. Application data can be saved as images or video files.
Client application was created for online monitoring by user. It communicates with security guard software over network by TCP/IP protocol. Implemented application layer protocol allows simple client authentication and data encryption.

Keywords

Linux, GTKmm, motion detection, video capture, V4L, video streaming, client authentication.

Citace

Jan Valeš: Zabezpečení prostoru pomocí videokamery a OS Linux diplomová práce, Brno, FIT VUT v Brně, 2007

Zabezpečení prostoru pomocí videokamery a OS Linux

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jaroslava Škarvady.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Bc. Jan Valeš
V Brně, 22.4.2007

Poděkování

Děkuji panu Ing. Jaroslavu Škarvadovi za jeho vedení a podnětné návrhy na vylepšení této práce.

Obsah

1 Úvod

V posledních letech jsme svědky rozšiřování technologií, dříve určených pouze pro omezenou skupinu profesionálů, mezi běžné uživatele výpočetní techniky. Mezi tyto technologie lze zařadit i kamery umožňující v reálném čase snímat obraz a přenášet ho do počítače. Ještě před několika roky byla kamera propojená s počítačem k vidění pouze na specializovaných pracovištích vybavených výkonnými servery schopnými digitalizovat a komprimovat příchozí data. Jednalo se o klasické kamery připojené přes analogové konektory k digitalizačním zařízením.

S pokrokem ve výrobě CCD čipů a s příchodem technologií jako je USB či FireWire se situace změnila. Tyto rychlé sběrnice umožnily digitální obraz ze snímačů přenášet v dostatečné kvalitě i do běžných osobních počítačů. Ve spojení s rostoucí rychlostí internetových připojení přinesla tato zařízení mimo jiné velmi zajímavou možnost provozovat videokonferencí. Tato nová forma komunikace byla hlavní příčinou rozšiřování tzv. web kamer mezi uživatele počítačů. Přestože interakce s jinými uživateli je jistě jednou z lákavých možností využití kamery propojené s počítačem, rozhodně ale není jedinou. Kamera může svému majiteli sloužit i v době, kdy zrovna není přítomen u počítače nebo dokonce ani ve svém obydlí. Za pomoci speciálního software je schopná vyhodnocovat snímaná data a monitorovat, co se děje v zabíraném prostoru (např. zjistit zda se v místnosti děje něco podezřelého např. pohybující se osoba, která zde nemá co dělat atd.)

Kamera tak slouží jako strážce schopný upozornit svého majitele na nestandardní situaci přímo v době jejího vzniku. Ten může okamžitě provést kroky nezbytné k vyřešení případného problému. Upozornění může mít formu třeba emailu nebo SMS či MMS zprávy, pokud ji lze zaslat přes internet nebo je připojena mobilní stanice.

Další možností je uložit detekovanou akci na disk ve formě video souboru, aby si uživatel mohl prohlédnout co se v době jeho nepřítomnosti ve sledovaném prostoru dělo. Výhodou je možnost přehrávání záznamů přes internet, případně možnost sledovat dění před kamerou v reálném čase. Takové řešení sebou nese potřebu autentizace uživatele a zabezpečení přenosu. Nebylo by totiž vhodné, aby taková data mohl prohlížet kdokoli na internetu. Například i potencionální zloděj, který by si obhlédl co může zcizit a kdy je k tomu nejvhodnější doba.

Problémem na cestě k přesnější detekci pohybu v obraze je kvalita snímaného obrazu. Šum a špatné osvětlení snímané scény mohou detekování důležitých událostí znesnadnit. V případě venkovního použití je situace ještě obtížnější vlivem povětrnostních podmínek. Husté sněžení, kapka vody na čočce kamery nebo pohyb objektů způsobený poryvy větru jsou velkou překážkou k detekci požadovaného druhu pohybu na scéně. V takových případech je nutno použít sofistikovanější metody. Jaké by tyto metody měly být? Právě na takové a další otázky hledá tato práce nejlepší odpovědi a jejím cílem je implementovat řešení, která by svým uživatelům přinesla nejvyšší možnou flexibilitu, rychlost a bezpečnost.

2 Aplikační programování pod Linuxem

2.1 Linuxový démon

Slovem démon se označuje proces běžící v pozadí, který je vytvořen pro samostatný běh bez přímého ovládání nebo zásahů od uživatele. Jeho úkolem je poskytovat systémové služby jako jsou tisk, plánování uživatelských úloh, HTTP server a jiné. Je napsán tak, aby spotřebovával co nejméně systémových zdrojů, dokud nejsou explicitně požadovány jeho služby. Démon běží často s root právy, ale je možné i spuštění s právy specifického účtu. Ekvivalentem v operačních systémech firmy Microsoft je mu service, česky systémová služba.

Výhoda, kterou použití démona přináší, je také v tom, že ostatní aplikace mohou využívat jeho služby a nemusí mít kompletní znalosti o konkrétním zařízení nebo práva pro jeho přímou obsluhu. Příkladem může být požadavek na webovou stránku. Klientská aplikace pouze vytvoří dotaz na požadovaná data a nestará se o to, jak je získat. Požadavkem je probuzen démon http serveru, který ověří přístupová práva, vyhledá na fyzickém disku požadovaný dokument a vrátí ho klientské aplikaci.

Démoni jsou většinou spouštěni systémem při jeho startu a existují po celou dobu jeho běhu. Jsou základními stavebními kameny Linuxu a právě díky nim poskytuje tento systém takovou výkonnost a flexibilitu.

2.1.1 Tvorba démona

Vytvoření démona pod Linuxem se řídí množinou pravidel uspořádaných v danou posloupnost. Díky tomu mohou démoni fungovat jak v uživatelském módu, tak i provádět volání modulů jádra, pracujících s hardwarovými zařízeními jako jsou síťové karty, externí adaptéry, tiskárny a v neposlední řadě i kamery.

Existuje nepsaná konvence, že názvy programů, které mají být démony končí písmenem "d". Např. httpd, crond, lpd a další. Doporučuje se, aby démon nikdy nekomunikoval přímo s uživatelem přes terminál ani jinak. Veškerá komunikace by měla být zajištěna přes zasílání signálu nebo síťovou komunikaci, grafické uživatelské rozhraní případně konfigurační soubory a LOG soubory.

Při spuštění musí démon provést několik akcí než začne plnit svoji funkci. Dle linuxquestions.org [2] těmito akcemi jsou:

- Oddělení od rodičovského procesu - ať už je démon spuštěn systémem, skriptem nebo uživatelem je potřeba ho udělat plně nezávislým
- Použít funkci umask - pokud chceme pracovat se soubory generovanými démonem musíme nastavit souborovou masku (umask(0)).
- Otevřít log pro zápis - nepovinný krok, ale vhodný pro ladění a kontrolu funkce
- Vytvořit unikátní Session ID - pro to aby se z procesu nestal po zániku otcovského procesu sirotek.
- Změnit pracovní adresář na bezpečný - nastavit pracovní adresář na takový, který jistě existuje. (Jediný zaručený je kořenový adresář)
- Zavřít standardní vstupy/výstupy - démon nepracuje s terminálovým rozhraním, jsou zbytečné a navíc vnášejí možnost bezpečnostních rizik.
- Vytvořit nekonečnou smyčku - ta vykonává funkci pro niž byl démon vytvořen.

2.1.2 Komunikace s okolím

Démon si může ostatními běžícími procesy vyměňovat data nebo se synchronizovat pomocí standardních prostředků meziprocesové komunikace. Jsou to signály (signal), roury (pipe), pojmenované roury (named pipe), sokety (socket), sdílená paměť (shared memory), semafory (semaphore), zasílání zpráv (message passing), soubor mapovaný do paměti zpráv (message queue).

Signály se dají použít i pro omezené ovládání démona uživatelem. Například pokud dojde ke změně některých parametrů v konfiguračním souboru je možné zasláním signálu třeba z příkazové řádky (např. signálu SIGUSR 1) vynutit znovunačtení parametrů z konfiguračního souboru.

Pro komunikaci s dalšími vlákny nebo procesy běžícími na stejném stroji, těsně spojenými s démonem a jeho činností se hodí prostředky jako je sdílená paměť nebo soubor mapovaný do paměti. Příkladem může být okno se zobrazením video dat, která démon snímá z kamery a dále používá pro detekci pohybu.

Hlavním prostředkem pro distribuované systémy, je princip komunikace klient - server. Pro takový typ komunikace nejlépe vyhovuje použití socketů. Otázkou zůstává pouze, zda zvolit rychlou komunikaci nespojovaným protokolem UDP nebo je možné si dovolit pomalejší přenos spojovanou službou TCP.

2.2 Knihovny pro tvorbu GUI

Při tvorbě dnešních aplikací si programátor jen těžko vystačí pouze s příkazovou řádkou. Především v to platí případech, kdy je hlavním úkolem daného programu zobrazovat obrazová data, ať již statická či dynamická. Jelikož je součástí tohoto projektu i vytvoření klientské aplikace pro zobrazování nahraného nebo snímaného videa, je nutné vytvořit uživatelsky přívětivé rozhraní.

Pro tvorbu GUI je možné využít z některou z množství již existujících grafických knihoven. Poskytují předdefinované prvky pro výstavbu uživatelských oken a jejich funkčnost je již prověřená. Někdy jsou na nich postavena i celá desktopová prostředí. Tyto knihovny a nástroje s nimi distribuované se označují jako Toolkit. V souvislosti s programováním grafických uživatelských rozhraní se často vyskytuje termín Widgets (Controls). Český překlad pod OS Windows je ovládací prvky. Jedná se o objekty použitelné k vytvoření oken aplikace. Většinou jsou všechny odvozeny od jednoho objektu a dědí jeho základní atributy a metody. Následující toolkity patří k těm nejpoužívanějším při vývoji GUI aplikací proto, jsou zde shrnuty jejich základní vlastnosti.

2.2.1 GTK+

GTK je zkratkou sestavenou z The GIMP Toolkit. Vzniklo totiž jako grafický toolkit pro editor obrázků GIMP. Jedná se o jeden z nejpopulárnějších nástrojů pro tvorbu grafických uživatelských rozhraní pro X Window Systém. Mimo jiné ho využívá i desktopové prostředí GNOME.

Toolkit GTK+ je napsán v jazyce C. Protože je objektově orientovaný, bylo potřeba v C implementovat prostředky pro manipulaci s objekty, které jsou v jazycích jako C++ součástí jazyka. Třídy jsou definovány jako struktury, odvozená třída obsahuje strukturu bázové třídy jako svou první položku. Virtuální metody jsou realizovány pomocí signálů. Zde je třeba odlišit unixové signály (asynchronní) a signály GTK+, které poskytují mechanismus pro synchronní volání registrovaných handlerů. V C++ lze použít buď C-čkové rozhraní, nebo knihovnu gtkmm (dříve GTK--). Jednotlivé třídy GTK+ jsou v gtkmm „obaleny“ třídami jazyka C++. Grafické rozhraní programu se definuje buď přímo v kódu programu posloupností volání funkcí na vytvoření jednotlivých widgetů, nebo lze použít nástroj Glade pro vizuální interaktivní návrh vzhledu aplikace. Programátor myší umísťuje jednotlivé widgety na obrazovku a nastavuje jejich parametry. Glade následně vygeneruje kostru aplikace obsahující kód na vytvoření widgetů. Do této kostry je pak nutné doplnit zbytek programu, především těla handlerů signálů [4].

Pro GTK+ existuje aplikační rozhraní pro mnoho dalších programovacích jazyků a prostředí jako je C++, Java, Perl, Python, .NET (MS Windows), Ruby. Aplikace napsaná v GTK+ je tedy dobře přenositelná mezi různými systémy, což je podstatná výhoda. Další výhodou je LGPL licence umožňující využití i v uzavřených programech.

Spolu s GTK+ jsou distribuovány další knihovny např. GLIB (knihovna pro práci s pamětí, seznamy, stromy, řetězci a mnoha dalšími), Pango (knihovna starající se o vykreslování textu), Cairo (knihovna na práci s vektorovou grafikou), GDK (mezivrstva mezi GTK+ a Xlib)

Pro srovnání jednotlivých toolkitů obsahuje každý z nich jednoduchý příklad vypsání dnes již klasické hlášky Hello world!

Příklad č.1 Hello World s GTK+:

```
#include <gtk/gtk.h>

int main( int   argc,
          char *argv[] )
{
    GtkWidget *window;
    GtkWidget *label;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    label = gtk_label_new("Hello World!!!");
    g_signal_connect (G_OBJECT (window), "delete_event",
  G_CALLBACK (gtk_main_quit), NULL);
    gtk_container_add (GTK_CONTAINER (window), label);
    gtk_widget_show_all(window);

    gtk_main ();

    return 0;
}
```

2.2.2 Qt

Za multiplatformní knihovnou Qt stojí firma Trolltech. Jedná se spolu s GTK+ o nejvýznamnější toolkity pro tvorbu GUI. Qt je objektově orientované a napsané v C++ rozšířeném o mechanismy definice signálů a slotů, dynamické identifikace typů a vlastností. Rozšíření jsou implementována pomocí maker a preprocesoru moc (Meta Object Compiler)[4]. Qt je použita pro desktopové prostředí KDE.

Qt je možné použít na mnoha platformách zahrnujících Linux, Windows, Mac OS X i embeded systémy jako jsou Smartphone, PDA a další. Existuje i verze pro Javu a tak může běžet i na dalších systémech pro která je vytvořena implementace Java Virtual Machine

Qt používá vlastní vykreslovací jádro a widgety. To umožňovalo lehčí portabilitu mezi platformami, protože většina tříd nebyla systémově závislá na cílové platformě. Qt se snažilo emulovat standardní vzhled systému na němž běželo, což se ne vždy zcela podařilo. V současné době ale tento problém ustupuje, neboť poslední verze Qt používají nativní API různých platforem pro vykreslování ovládacích prvků.

Existuje několik licencí pro Toolkit Qt. Je dostupný buď pod komerční licencí pro vývoj uzavřených aplikací nebo pod GPL, popř. QPL (Qt Public License) pro vývoj freeware. Oficiální

verze pro OS Windows je pouze komerční, existuje však projekt Q../Free, přinášející GPL licencovanou verzi.

Mezi další funkce toolkitu patří přístup do SQL databází, zpracování XML souborů, správa vláken, a jednotné mezi platformní rozhraní pro práci se soubory.

Příklad č.2 Hello World s Qt:

```
#include <qapplication.h>
#include <qpushbutton.h>


int main( int argc, char **argv )
{
    QApplication a( argc, argv );

    QPushButton hello( "Hello world!", 0 );
    hello.resize( 100, 30 );

    a.setMainWidget( &hello );
    hello.show();
    return a.exec();
}
```

2.2.3 FLTK

Název je zkratkou slov "Fast, Light Toolkit" (vyslovuje se "fulltick") a plně vystihuje hlavní vlastnosti tohoto Toolkitu. Byl vytvořen Billem Spitzakem pro programování 3D grafiky a obsahuje interface pro OpenGL. FLTK knihovna je určena pro vývoj aplikací v C++. Programy vytvořené pomocí FLTK vypadají stejně na všech podporovaných operačních systémech.

Na rozdíl od předešlých toolkitů obsahuje FLTK pouze funkce týkající se tvorby GUI. Díky tomu je knihovna poměrně malá a je možné ji linkovat staticky. To je její největší výhoda.

Existují implementace pod Linux, BSD (X11), Mac OS X a Windows. Toolkit je šířen pod GPL licencí a obsahuje program fluid (FLTK User Interface Designer) určený pro grafické vytváření GUI.

Příklad č.3 Hello World s FLTK:

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>

int main(int argc, int **argv) {
  Fl_Window *window = new Fl_Window(300,180);
  Fl_Box *box = new Fl_Box(20,40,260,100,"Hello, World!");
  box->box(FL_UP_BOX);
  box->labelsize(36);
  window->end();
  window->show(argc, argv);
  return Fl::run();
}
```

2.2.4 wxWidgets (wxWindows)

Dalším mezi platformním toolkitem obsahujícím základní prvky pro stavbu grafických uživatelských rozhraní je wxWidgets (Windows and X widgets), též známé jako wxWindows.

Jedná se o open source projekt šířený pod licencí podobnou LGPL. Rozdílem je, že produkty z něho vycházející, distribuované ve formě binárních souborů by mely obsahovat vlastní licenční ujednání

Otcem projektu vyvíjeného od roku 1992 je Julian Smart, který je stále jedním z vývojářů.

Knihovna je naimplementována v C++, ale existují možnost vazby na C#/.NET, Python, Perl, Ruby, Smalltalk, Javu, Lisp a mnohé další. Kompletní výčet je možné shlédnout na stránkách projektu.

wxWidgets umožňují, aplikaci zkompilovat a spustit na podporovaných platformách a minimálními nebo žádnými zásahy do kódu. Podporovány jsou operační systémy jako Windows, Apple Macintosh, Linux/Unix, OpenVMS a OS/2. Ve vývoji je i verze pro embeded zařízení.

wxWidgets je založen na použití tenké abstraktní vrstvy, odstiňující implementační detaily jednotlivých systémů. Tato vrstva se nesnaží pomocí primitiv vytvářet vlastní widgety, ale využívá nativních ovládacích prvků konkrétního systému. To přináší zrychlení běhu a především design aplikace lépe zapadající do cílového prostředí.

Mezi další funkce wxWidgets patří podpora ODBC databázových spojení, vrstva pro meziprocesovou komunikaci, funkce pro práci se síťovými sokety, funkce pro generování HTML nebo podporu OpenGL.

Příklad č.4 Hello World s wxWidgets:

```
#include "wx/wx.h"

class MyApp: public wxApp
{
   virtual bool OnInit();
};

class MyFrame: public wxFrame
{
   public: MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size);
};

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
   MyFrame *frame = new MyFrame( "Hello World", wxPoint(50,50), wxSize(450,340) );
   frame->Show(TRUE);
   SetTopWindow(frame);
return TRUE;
}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
: wxFrame((wxFrame *)NULL, -1, title, pos, size)
{
}
```

2.3 Síťová komunikace

Existují standardy RFC (RFC 768 , RFC 793, RFC 1180,…) specifikující jaké funkce musí implementace protokolů rodiny TCP/IP poskytovat. V těchto dokumentech se však nevyskytují žádné definice toho, jak se má k protokolovému zásobníku přistupovat z procesů, které chtějí síťovou komunikaci využívat. Důvodem pro to je rozdílnost cílových platforem na kterých má být protokol provozován. Přes tyto rozdíly vzniklo několik typů rozhraní jenž se postupně stala nepsanými standardy a jsou využívána při tvorbě nových systémů. Mezi takové vzory patří i BSD sokety.

Všechna aplikační síťová rozhraní, podporující rodinu protokolů TCP/IP, musí implementovat jistou sadu funkcí, které můžeme dle článku Rozhraní BSD Sockets [5] shrnout do následujících bodů:

- alokace lokálních zdrojů nutných pro komunikaci
- specifikace lokálního a vzdáleného komunikačního uzlu
- zřizování spojení - klient
- čekání na požadavek spojení - server
- čtení a zápis dat
- indikace příchodu dat
- zpracování urgentních zpráv
- aktivní a pasivní ukončení komunikace, násilné ukončení komunikace
- zpracování chybových stavů

2.3.1 BSD Sokety

Při vytváření návrhu komunikačního modelu implementujícího protokol TCP/IP pro operační systém BSD Unix se tvůrci rozhodli vypracovat obecnější řešení, které by mohlo podporovat širší okruh protokolů, dokonce i ty které do rodiny TCP/IP přímo nepatří. Takové řešení má výhodu ve své snadné rozšiřitelnosti rozhraní o implementace nově definovaných protokolů. Toto rozhrání zpřístupňující služby transportních protokolů je označováno jako socket interface, zkráceně sockets - sokety.

Na druhou stranu univerzálnost návrhu přináší i komplikace. Zejména se jedná o složitější použití. Programátor aplikace musí nastavit datové struktury popisující jakým protokolem a jakým způsobem se chystá komunikovat. Kdyby se jednalo o řešení připravené přesně pro TCP/IP stačilo by pro inicializaci komunikace funkci předat pouze IP adresu a port. Tuto nevýhodu lze snadno odstranit například napsáním tříd obalujících struktury a metod volajících standardní funkce knihoven.

2.3.1.1 Struktura BSD socketů

Pod operačními systémy Unix/Linux je zvykem pracovat s koncovými zařízeními, jakým socket v přeneseném slova smyslu bezesporu je, jako se soubory. Rozhraní BSD Sockets vychází z tohoto konceptu a rozšiřuje ho tak, aby vyhovoval jak funkcím určeným pro práci se soubory tak i pro komunikaci prostřednictvím sítě. Síťová komunikace má oproti práci se soubory svá specifika a proto je nutné připojit sadu nových funkcí.

Jedním z hlavních parametrů soketu je tzv. rodina protokolů (protocol family) například rodina PF_INET slouží pro použití v Internetu a zahrnuje protokoly IP, TCP a UDP.

Pro každou rodinu protokolů existuje jedna nebo více rodin adres (address family). V internetu se pro komunikaci pomocí 16-bytových adres používá rodina AF_INET. Protože se pod konstantou pojmenovanou AF_INET skrývá stejná hodnota jako pod PF_INET dochází často k jejich nesprávnému zaměňování.

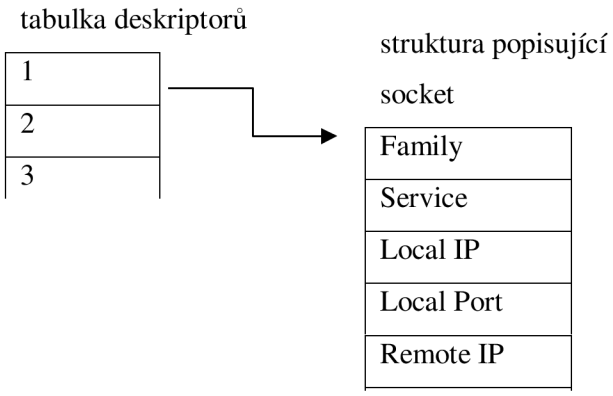
Další vlastností rozhraní BSD Sockets je způsob komunikace. Jedná se buď o virtuální spojení (SOCK_STREAM) nebo datagramovou službu (SOCK_DGRAM).

Virtuální spojení poskytuje plně duplexní, spolehlivý kanál, který zaručí že data odeslaná jednou stranou budou v nezměněné podobě doručena straně druhé. Cenou za to je paměťová režie ve formě bufferů a teoreticky pomalejší přenos způsobený žádáním o opakované zaslání ztracených či poškozených dat. V prostředí internetu odpovídá použití protokolu TCP.

U datagramové služby se jedná o tzv. nespojovanou službu. Souží pouze k odeslání dat. Nezaručuje jejich doručení ve správném pořadí, neduplikovaných a neposkytuje mechanizmy žádosti o porušená či ztracená data. Výhodou je nižší režie a teoreticky i větší propustnost. Takovou službu zprostředkovává protokol UDP.

Analogicky práci se souborem je před prací se soketem nutné ho nejprve otevřít. Samotné otevření provede jádro systému po systémovém volání z procesu požadujícího socket. Zde je nutné specifikovat jak bude socket vypadat a jaké bude jeho chování.

Pokud skončí vytvoření soketu úspěchem, je mu přiřazen identifikátor z tabulky deskriptorů stejně jako je tomu v případě nově otevřeného souboru. Hodnoty se vybírají ze stejné množiny jako je tomu u souborů, nemůže se tedy stát, že by v systému byl soubor a socket se stejným identifikátorem. Systém vytvoří pomocnou datovou strukturu popisující vlastnosti soketu (viz. Obrázek 1.) a vrátí volajícímu procesu jeho deskriptor. Proces skrze tento deskriptor může ze soketu číst nebo do něj zapisovat.



Obrázek 1: Struktura soketu

Podle způsobu otevření lze rozlišovat TCP sokety na aktivní a pasivní. Toto označení vychází z dokumentu RFC 793.

2.3.1.2 Režimy BSD socketů

Pasivní socket je takový, který vyčkává na požadavek na příchozí spojení a po jeho příchodu na něj odpovídá. Toto je základní vlastnost serveru poskytujícího na síti své služby. Aktivní je naopak takový socket, který sám zahájí komunikaci tím, že zašle požadavek na adresu pasivního soketu. Takové chování umožňuje klientům vytvářejícím připojení k serveru, aby využili poskytovaných služeb a zdrojů.

Existují situace jako je navazování spojení nebo čtení příchozích dat, jejichž vyřízení může vyžadovat poměrně dlouhý čas. Ne vždy je vhodné, aby takové čekání zablokovalo běh aplikace. Proto je možné aby byl socket nastaven buď do blokujícího nebo neblokujícího módu. Každý socket je po vytvoření implicitně nastaven do blokujícího režimu.

Pokud je socket nastaven do neblokujícího režimu a dojde k volání funkce, jež nemůže být okamžitě provedena vrátí funkce řízení zpět procesu se zvláštním chybovým kódem (EWOULDBLOCK).

Jako existují blokující a neblokující módy socketů je možné volat blokující a neblokující funkce. Voláním blokující funkce na blokujícím soketu může být běh celé aplikace zablokován na předem nestanovenou dobu. Pokud druhá strana spojení z nějakého důvodu neodpoví nastane deadlock a proces není schopen dalšího běhu. Oproti tomu volání neblokující funkce na blokujícím soketu je bezpečné, neboť tyto funkce vracejí řízení procesu okamžitě.

2.4 POSIX Threads

Termínem vlákno se označuje část kódu, většinou funkce nebo několik funkcí, běžících nezávisle a paralelně v rámci jednoho procesu. Vlákna mohou existovat pouze v rámci nějakého procesu. Pokud je proces ve kterém byla vlákna vytvořena ukončen, jsou všechna jeho vlákna taktéž ukončena. Systém může plánovat běh jednotlivých vláken procesu naprosto nezávisle na ostatních
Vlákna mají oproti procesům několik podstatných rozdílů [9].

Vlákna mají společné:

- instrukce programu
- globální proměnné
- otevřené deskriptory
- signály a jejich obslužné rutiny

Každé vlákno má vlastní:

- identifikátor vlákna (Thread ID)
- hodnoty registrů včetně ukazatele zásobníku
- zásobník
- prioritu
- lokální proměnné
- chybovou proměnnou errno

Jelikož jednotlivá vlákna procesu sdílejí mnohá data i mapováni fyzické paměti na virtuální, je přepnutí kontextu mezi takovými vlákny mnohem rychlejší než by tomu bylo u přepínání mezi procesy. To přináší podstatné zrychlení paralelně pracujících algoritmů.

Další výhodou je jednoduchá komunikace mezi jednotlivými vlákny, která může být implementována přes jednoduše alokovatelnou sdílenou paměť. U procesů se nevyhneme použití některé z forem meziprocesové komunikace, jenž byly představeny v kapitole 2.1

Otázkou je jak vlákna vytvářet a provádět jejich kontrolu a správu za běhu. Rozhraní knihovny definující datové typy a funkce pod operačním systémem UNIX bylo specifikováno již v roce 1995 ve standardu IEEE POSIX 1003.1c.

Při využití POSIX vláken v praxi narážíme na potíže. Ty jsou způsobeny různými verzemi standardů podle nichž jsou jednotlivé verze implementovány.

2.4.1 Synchronizace

Jak již bylo řečeno, vlastností kterou se vlákna odlišují od procesů je sdílení množiny zdrojů a dat. To co se dříve jevilo jako podstatná výhoda se sebou přináší i některé obtíže. Kvůli tomu, že operační systém může plánovat přidělování procesorového času vláknům nezávisle, je třeba počítat s předem neurčeným pořadím běhu jednotlivých vláken procesu.

Situaci je možné demonstrovat na praktickém příkladu, kdy jedno z vláken obstarává získání obrazu z kamery, jeho předzpracování a detekci pohybu. Další vlákno nebo jejich skupina zajišťuje streaming zpracovaných dat klientům naprogramovaných pro zobrazování videa. Činnost vlákna může být kdykoli systémem pozastavena a jiné vlákno spuštěno. Předpokládejme, že vlákna pracují nad jedním paměťovým bufferem, přes který si předávají data. Zde je nutné zabránit posloupnosti provádění kódu, jež by měla za následek přerušení zpracovávání obrazu ve chvíli, kdy by do bufferu ještě nebyla umístěna celá nová hodnota, a následné odeslání neúplných dat klientům. Proces zabraňující vzniku takových nekonzistencí, označovaných jako chyby souběhu, se nazývá synchronizace.

Knihovny pracující s vlákny naštěstí poskytují hned několik možností jak se podobným nepříjemnostem vyhnout. Jsou to:

Mutex - Jedná se o nejjednodušší formu synchronizace vláken. Pracuje na principu nastavování zámku na globálně definovaných strukturách. Mutex může být zamknut maximálně jedním vláknem a ostatní musí čekat až vlákno provede jeho odemčení. Mutex se vytváří voláním, k zamknutí se používá funkce pthread_mutex_lock(), na odemknutí pthread_mutex_unlock() a další.

Příklad č.5 Mutex:

```
pthread_mutex_init(&mut, NULL);

Vlákno 1                          Vlákno 2
pthread_mutex_lock (&mut);
                                  pthread_mutex_lock (&mut);
napln_buffer(&buff);              /* blokováno */
full=1;                           /* blokováno */
pthread_mutex_unlock (&mut);      /* blokováno */
                                  zpracuj_buffer(&buff);
                                  full=0;
                                  pthread_mutex_unlock (&mut);
```

Podmínkové proměnné (Condition variables) - Ke své funkci využívají Mutexů a zasílání signálů. Je nutné definovat proměnou jenž má být synchronizována, mutex strukturu podmínkové proměnné. Oproti mutexům přidávají výhodu pasivního čekání na zpracovaná data. Vlákno A zavolá pthread_cond_wait() a pozastaví vykonávání. V momentě, kdy vlákno B data zpracuje tak, že je splněna podmínka, zavolá pthread_cond_signal(), to zašle signál a tím probudí vlákno A. Tento postup šetří čas procesoru, neboť není nutné proměnou opakovaně testovat na změnu hodnoty.

Je možné vytvořit i časově omezené čekání. Pokud v takovém případě není podmínka splněna do stanoveného intervalu vrací funkce speciální chybovou hodnotu a pokračuje se v provádění kódu.

Příklad č.6 Podmínkové proměnné:

```
pthread_mutex_t mutex;
pthread_cond_t data_ready;

Vlákno 1
pthread_mutex_lock(&mutex);
pthread_cond_wait(&data_ready, &mutex); //automaticky uvolni mutex a po
 //návratu zamkne
send_data();
pthread_mutex_unlock(&mutex);

Vlákno2
fot(int x=0; x<=WIDTH; x++)
{
	pthread_mutex_lock(&mutex);
	process_row(x);
	if (x == WIDTH)
	{
		pthread_cond_signal(&data_ready);
		printf("Radka zpracovana\n");
	}
	pthread_mutex_unlock(&mutex);
}
```

Semafory - V případě semaforu se jedná o synchronizaci na principu podobném Mutexu. Rozdíl je v tom, že u semaforů je možné provést i více než jedno zamčení. Semafor tak může efektivně sloužit k simulaci omezeného množství zdrojů. Po každém zamčení se dekrementuje čítač. Pokud dosáhne čítač nuly je semafor uzamčen a volající funkce čeká až se někdo jiný semaforu vzdá.

Příklad č.7 Semafor:

```
void *thread_function(void *arg)
{
	while(1)
	{
		sem_wait(&job_queue_count);
		pthread_mutex_lock(&job_queue_mutex);
		next_job = job_queue;
		job_queue = job_queue->next;
		pthread_mutex_unlock(&job_queue_mutex);
		process_job(next_job);
		free(next_job);
	}
	return(NULL);
}
```

2.5 Video4Linux

2.5.1 Historie

Video4Linux je aplikační rozhraní linuxového jádra sloužící pro práci se zařízeními pro zachytávání obrazu nebo jeho vykreslování. Jeho první verze se stala standardní součástí jádra od verze 2.1. Aby bylo možné zařízení používat musí mít nainstalován v systému V4L ovladač. Potom API V4L umožňuje práci s periferií bez znalosti hardwarového designu nebo detailů jejich ovladačů. Pomocí V4L lze pracovat například s kartami pro digitalizaci videa, TV kartami, FM kartami či videokamerami.

Brzy poté, co bylo V4L přidáno do jádra se objevila kritika zaměřená na jeho malou flexibilitu. Na podzim roku 1998 navrhl Bill Dirks mnohá vylepšení a začal pracovat na vzorových příkladech ovladačů a aplikací. Spolu se skupinou dobrovolníků vytvořil V4L2 API, které se místo rozšíření stalo přímo náhradou starého V4L. Trvalo však další čtyři roky než bylo přidáno jako pevná součást jádra a to ve verzi 2.5. Stále se však v praxi setkáváme s mnohými ovladači pracujícími se specifikací V4L a proto je nutné, mít jeho podporu buď přímo nebo skrze vrstvu zpětné kompatibility ve V4L2.

2.5.2 Specifikace Video4Linux2

Popis specifikace API by sám o sobě mohl vydat na samostatnou publikaci, s příklady kódu aplikací a ovladačů na dvě. Toto téma se přímo netýká této práce a proto zde bude uveden jen stručný přehled. Více v Video for Linux Two API Specification [11]. Programování aplikací využívajících V4L2 zařízení se skládá z:

- Otevření zařízení
- Nastavení parametrů zařízení a obrazu, vybrání audio/video výstupů
- Vyjednání datového formátu
- Vyjednání vstupních/výstupních metod
- Vlastní čtení/zápis dat
- Zavření zařízení

V4L2 ovladače jsou implementovány jako moduly jádra. Ovladače se připojují k "videodev" modulu jádra poskytujícímu standardní rozhraní. Ovladač V4L zařízení je identifikován majoritním číslem 81 a minoritním v rozsahu 0-255. Každé zařízení musí mít unikátní minoritní číslo aby nedocházelo k vzájemným konfliktům. Pokud jedno zařízení poskytuje více funkcí, je mu přiděleno pro každou funkci jedno minoritní číslo. To platí i pro V4L2, kvůli zpětné kompatibilitě s předešlou verzí.

Nicméně je ve V4L2 možné otevřít některé z fyzické zařízení a za běhu se přepínat mezi jeho virtuálními s podobnou funkcí.

Zdroj Video for Linux Two API Specification [11] jako příklad uvádí ovladač podporující zachytávání videa, zobrazování videa, zachytávání VBI (teletext, časové kódy a další data v TV signálu) a příjem FM rádia. V takovém případě jsou zaregistrovány tři zařízení s minoritními čísly 0, 1 a 64 (číslovací schéma zděděné z V4L API). Bez ohledu na to zda je otevřeno zařízení /dev/video (81, 0) nebo /dev/vbi (81, 1) je aplikace schopna si vybrat jednu z funkcí zachytávání videa, zachytávání VBI nebo zobrazování. Pod /dev/radio (81, 64) je umístěno zařízení schopné přijímat rádio, bez podpory video funkcí.

Rozšířením V4L2 oproti svému předchůdci je možnost vícenásobného otevření zařízení. Ovšem pouze tehdy, pokud je to podporováno ovladačem a samotné zařízení je fyzicky schopné vykonávat více činností ve stejnou dobu. Jedna aplikace se pak může starat o přehrávání dat z integrovaného FM přijímače a další o zobrazování nebo ukládání dat zachycených TV tunerem.

V4L2 ovladače nepodporují současné čtení nebo zápis z více aplikací do jednoho datového bufferu nebo jeho časové sdílení. Pokud je taková funkce vyžadována je nutné ji implementovat službou na úrovni uživatelské aplikace.

2.5.3 Komunikace

Pro komunikaci se zařízením se využívá systémové funkce ioctl a jejích variant s parametry. První, se kterým se při vývoji aplikací s podporou V4L2 programátor setká je VIDIOC_QUERYCAP. Protože V4L2 podporuje velkou škálu nejrůznějších periferií je třeba zjistit jaké metody zařízení podporuje a zda je vůbec se standardem kompatibilní (u V4L tuto funkci zabezpečoval VIDIOCGCAP). Další informace o zařízení je možné získat voláním VIDIOC_ENUMINPUT.

Pokud je u V4L2 řeč o video vstupech/výstupech, jedná se fyzické konektory na zařízení jako jsou S-video, D-SUB, kompozitní a jiné. Zařízení pro zachytávání videa mají video vstupy a zařízení pro zobrazování video výstupy. Před použitím je nutné nastavit požadovaný vstup/výstup pomocí ioctl a VIDIOC_S_INPUT/VIDIOC_S_OUTPUT.

Podobně jako mají zařízení video vstupy/výstupy (V/V) mají i audio V/V. V případě V4L2 API jsou audio V/V asociovány s video V/V. Nastavení určitého parametru pro video se tedy projeví i u audia. Toto audio nemá nic společného s FM rádiem a podobnými zařízeními neobsahujícími video V/V, ke kterým by mohly být audio V/V asociovány. API poskytuje dále mnoho funkcí pro nastavení tunerů, modulací i video standardů. Podrobný popis je k nalezení na stránkách věnovaných popisu rozhraní ve Video for Linux Two API Specification [11].

Jednotlivá zařízení jsou schopna posílat aplikaci rozdílná data a proto před zahájením komunikace se zařízením dochází k vyjednávání datového formátu. Přestože datový formát V/V dat zůstává persistentní při zavírání a znovu otevírání zařízení, je před každou komunikací doporučeno

provést vyjednání formátu pro výměnu dat. Vyjednává ní spočívá v zaslání požadavku na určitý formát a driver odpoví zda jej může v hardware uspokojit. U vícenásobného přístupu je pouze aplikace, která otevřela deskriptor a provedla vyjednávání, schopna změnit formát přenášených dat.

U V4L2 existuje několik způsobů jak provést samotnou komunikaci se zařízením. Drivery musí podporovat alespoň jednu z nich.

- Čtení/zápis – jedná se o jediné metody použitelné hned po otevření zařízení. Pokud je driver nepodporuje dojde při jejich volání vždy k selhání. Jejich implementaci lze ověřit voláním ioctl s parametrem VIDIOC_QUERYCAP. Ostatní metody musí být se zařízením vyjednány.
- Streaming - Mapování paměti – U streaming metod nedochází mezi aplikací a ovladačem ke kopírování dat, ale pouze ukazatelů. Mapování paměti využívá namapování bufferů v zařízení do uživatelské paměti.
- Streaming - Uživatelské ukazatele – Metoda kombinující výhody čtení/zápisu a mapování paměti. Dochází pouze k výměně ukazatelů.

3 Získání a zpracování obrazu

Definice webové kamery podle Wikipedie [13] je: "Web kamera (nebo též webcam) je real-time kamera (většenou, ale ne vždy s jedná o kameru) jíž vytvořené obrázky jsou přístupné přes WWW, instantní messengery nebo PC aplikace pro video hovory".

Jak z převzaté definice plyne dva hlavní směry pro šíření dat z kamery směrem k uživatelům spočívají v nahrávaní obrázků či videa na WWW servery nebo v zasílání real-time proudu dat koncové uživatelské aplikaci.

Oba dva principy lze využít při použití kamery jako zabezpečovacího zařízení. Aplikace může kopírovat video soubory s detekovaným pohybem do určeného centrálního úložiště nebo rovnou odesílat video stream klientům.

Pro to, aby mohla aplikace data ze zařízení vůbec získat, jsou potřeba ovladače, které ho zpřístupní.

3.1 Ovladače

Operační systémy Linux se dodnes ještě neprosadily mezi tak široké masy uživatelů tak jak je tomu u OS Windows firmy Microsoft. Výrobce multimediálních zařízení tím pádem nejsou výrazněji motivováni k vývoji ovladačů pro Linuxové systémy a sami v rámci úspory většinou žádné ovladače nevyvíjí. Proto je nutné využít ovladačů od třetích stran. Ty podporují určité skupiny výrobků, které spojuje do značné míry podobná nebo stejná hardwarová konstrukce. Mezi často používané ovladače patří:

- Philips USB Webcam Driver for Linux
- GSPCA / SPCA5xx Camera Linux Driver
- CPiA webcam driver for Linux
- ....

Další informace jsou k nalezení na stánkách Video for Linux resources [14].

Instalací ovladačů do systému se vytvoří v adresáři /dev soubory zařízení nazvané většinou videox, kde x je pořadové číslo. Přes tento soubor pak přistupují aplikace pomocí V4L API k zařízení. Rozdíly v jednotlivých ovladačích nejsou pouze v tom, jaká zařízení podporují ale též jakou verzi V4L API a do jaké míry implementují.

3.2 Snímání při nedostatku světla

Prvním problémem při získávání kvalitního obrazu je nedostatek světla ve snímané scéně. Příčinou nedostatku světla bývá většinou denní (resp. noční) doba nebo nedostatečné osvětlení místnosti. Řešení problému existuje několik.

Prvním řešením, bez použití externích prostředků, je nastavení delšího času expozice, pokud to kamera dovoluje. To přináší na druhou stranu problémy se vzrůstající hladinou šumu a menším počet snímků za sekundu. Snímky zachycující pohyb jsou pak většinou poměrně neostré.

Další řešení spočívá ve využití externího osvětlení scény. To lze provést buď za pomoci klasického světla nebo speciálních diod svítících v infračerveném spektru. Výhodou světla s viditelným spektrem je, že barevné podání scény odpovídá realitě. Oproti tomu infračervené světlo nedodává scéně barvu, ale je neviditelné pro lidského pozorovatele.

Díky CCD snímačům zpracovávajícím obraz je možné zachytit určitou část z infračerveného záření. Křemíkový CCD čip videokamer a digitálních fotoaparátů je v IR oblasti citlivější než lidské oko a detekuje záření až do vlnové délky cca 1,2 μm. Takové chování ale způsobuje odlišnosti od požadovaného barevného podání (u černobílého obrazu k velkému zkreslení nedochází). Proto jsou ve snímacích zařízeních IR filtry odstraňující co největší část takového záření. Přesto je stále možné jistou část spektra kamerou snímat.

3.3 Předzpracování

Předzpracováním obrazu se rozumí kroky provedené hned po získání dat ze zařízení před samotnou detekcí pohybu. Při počítačovém zpracování obrazu se v naprosté většině případů používá pouze intenzita v úrovních šedi a ne celý barevný obraz reprezentovaný červenou, modrou a zelenou složkou. K převodu se využívá přepočet podle rovnice zohledňující citlivost lidského oka na jednotlivé barevné složky.

$$I = 0{,}299 * R + 0{,}587 * G + 0{,}114 * B$$

Rovnice 1: Převod na úrovně šedi

Dalším krokem je výpočet jasového histogramu. Histogram je graf, který dává přehled o tom, kolik pixelů jednotlivých intenzit je na snímku obsaženo ve škále od nejtemnější do největšího jasu. Příklad histogramu obrazu s převahou tmavých bodů. je na Obrázku 2.

Pokud se většina bodů nachází v levé části histogramu je obrázek příliš tmavý, v případě že leží u pravého okraje je naopak přesvětlený. Tyto poznatky pak mohou sloužit jak k nastavení parametrů

kamery tak i k vylepšení již získaného obrazu na straně aplikace vynásobením hodnot určitým koeficientem.

Obrázek 2: Histogram tmavého obrázku

Pokud má aplikace za úkol i vylepšovat obraz pro lidského pozorovatele je dalším možným krokem aplikování algoritmů pro potlačení šumu v obraze. Obsah šumu v obraze je závislý na nastavení citlivosti snímače i na jeho celkové kvalitě.

Existuje více způsobů jak šum z obrázku odstranit. Ty jednoduché a rychlé mají nevýhodu v tom, že jejich použití vede ke ztrátě detailů v obraze a rozmazání hran. Na druhou stranu je možné je aplikovat i v reálném čase. Příkladem jednoduchých opatření k odstranění šumu jsou nízkofrekvenční filtry jako je průměrový filtr, gaussovský filtr, mediánový filtr apod.

3.4 Detekce pohybu

Na problematiku detekce pohybu lze nahlížet ze dvou pohledů. Prvním je pouhé zjištění, zda ve vstupních snímcích dochází ke změnám. Pokud počet změn přesáhne určitou úroveň, je v obraze detekován pohyb. Při takové způsobu detekce je nutným předpokladem, že pozice kamery se v čase nemění.

Druhým a mnohem složitějším přístupem je zjistit informaci o tom jaký objekt se v obraze hýbe. Příkladem je detekce obličeje, siluety nebo chůze. Takové řešení je mnohem náročnější na implementaci, na druhou stranu ale detekuje pouze ty objekty o něž máme zájem a nevyvolává poplach způsobený například povětrnostními podmínkami.

3.4.1 Detekce rozdílů

Nejjednodušší popis prvního způsobu přístupu k detekci pohybu je dán rovnicí:

$$\sum_{x,y} D(x,y) > T$$

Rovnice 2: Jednoduchá detekce rozdílů

kde D(x,y) = 0 pokud | Pixel[x,y]t-1 - Pixel[x,y]t | < ε a D(x,y) = 1 v případě, že

$|Pixel[x,y]t-1 - Pixel[x,y]t| \geq \varepsilon$

T – nejmenší počet změněných pixelů, který znamená pohyb.

Pixel[x,y]t-1 – hodnota intenzity na souřadnicích x,y v předchozím obrázku.

Pixel[x,y]t – hodnota intenzity na souřadnicích x,y v aktuálním obrázku.

ε – rozdíl intenzit o jaký se mohou lišit aby byly ještě považovány za stejný.

Pokročilejší varianta tohoto algoritmu pracuje tak, že rozdělí obraz do segmentů, většinou pomocí pravoúhlé mřížky a v každém segmentu zjišťuje zda se v něm nacházejí změny. Segment je označený jako změněný pokud je uvnitř více změn než stanovený práh T. To zabraňuje vzniku jedno či několika bodových rozdílů nezpůsobených pohybem, ale chybou při snímání. Takových chyb se dá samozřejmě zbavit i nízkofrekvenčními obrazovými filtry. Hustota mřížky a T pro segment udává nejmenší objekt, který je takto detekovatelný.

Nevýhodou výše popsané metody je značná citlivost na šum. Dále jsou jako pohyb detekovány i změny v intenzitě obrazu jako je rozsvícení světla, blesky při bouřce a další.

Dalším rozšířením algoritmu je nalezení shluků bodů které se změnily, tak lze ohraničit oblast, ve které se pohybující se objekt nachází. Taková informace se hodí buď pro vizuální zvýraznění místa, kde v obraze k pohybu dochází nebo tak získáme segment obrazu, který dále zpracuje "inteligentnější" metoda detekce.

Jevem, se kterým se tato metoda z principu nemůže vypořádat je pomalý pohyb. Pokud se objekt na scéně pohybuje velmi pomalu, nedochází ke změnám které by překročily nastavené prahy. Snížení hodnoty prahů by k řešení nevedlo, neboť by způsobilo mnoho planých poplachů.

Zmíněný problém je možné řešit tak, že aktuální obraz srovnáváme s uloženým vzorem (např. první sejmutý obrázek) nebo některým z posloupnosti dříve sejmutých snímků. Tím narůstají požadavky na systémové zdroje a to jak na paměť pro staré vzorky tak i výkon procesoru, neboť je nutné pro každý nový snímek provést dvě nebo i více porovnání.

Více informací k nalezení na Motion Detection Algorithms [15] nebo inspiraci na stránkách Michala Bláhy[21].

3.4.2 Pravděpodobnostní model

Další možný přístup jak získat informaci o objektech na scéně je vytvoření modelu udávajícího jaká je pravděpodobnost, že daný pixel je pozadím. To umožňuje pozadí odstranit. Teorií se zabývá např. Odečítání pozadí a sledování [16].

Model může být buď statický nebo dynamický. Statický model je vytvořen v učící fázi a k jeho vytvoření je třeba získat čistý obraz pozadí. Není adaptivní a neumí se vypořádat se změnami v intenzitě osvětlení.

Dynamický model se vyvíjí za běhu, takže se přizpůsobí aktuální scéně, ale jsou zde problémy spojené s nastavením učících parametrů.

Příkladem dynamického modelu pozadí je Staufferův Grimsonův algoritmus [17]. U této metody je model pozadí v určitém bodu dán směsí K gausiánů (většinou K=3-5).

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t} * \eta(X_t, \mu_i, \sigma_t^2)$$

Rovnice 3: Staufferův Grimsonův algoritmus

Kde Xt je bod X v čase t, ω je váha gausiánu, μ střední hodnota a σ rozptyl. Pro každý vstupní obrázek je v každém bodě zjišťováno zda odpovídá jednomu z gausiánů. Odpovídat zde znamená ležet v intervalu daném střední hodnotou a standardní odchylkou. Parametry nejlépe odpovídajících gausiánů jsou aktualizovány a jejich váhy zvýšeny:

$$\omega_{k,t} = (1 - \alpha) * \omega_{k,t-1} + \alpha * (M_{k,t})$$

Rovnice 4: Aktualizace vah

Mk,t = 0 pokud bod do intervalu nepadl a Mk,t = 1 pokud do intervalu padl. Pokud vstupní hodnota nepadne do žádného intervalu, je komponenta s nejmenší vahou nahrazena nově příchozí. Takto získané gausiány jsou seřazeny sestupně podle ω/σ a prvních B gausiánů, pro které platí rovnice 5 značí pozadí. T je práh určující citlivost [17].

$$B = argmin_b \left(\sum_{k=1}^{b} \omega_k > T \right)$$

Rovnice 5: Gausiány značící pozadí

4 Zabezpečení a přenos dat

4.1 Zabezpečení

Aplikace poskytující přístup k privátním informacím přes veřejnou síť potřebují implementovat metody, které umožní přístup k datům pouze oprávněným uživatelům. S touto problematikou jsou spojeny dva pojmy. Autentizace a autorizace.

Autentizací rozumíme proces jednoznačného ověřování osoby (subjektu), který vstupuje do systému. K autentizaci je nejčastěji užíváno jméno a heslo nebo certifikáty s veřejným a privátním klíčem.

Autorizací subjektu se rozumí proces ověření přístupových práv osoby (subjektu) vstupující do systému. Tento proces ve většině případů navazuje na proces autentizace. Uživateli jsou přidělena práva k užívání prostředků a zdrojů [18]. Přestože jsou tyto dvě činnosti nezávislé v mnoha případech bývají spojeny do jednoho procesu.

4.1.1 Autentizace

Nejběžnějším způsobem autentizace je zaslání uživatelského jména a hesla. Pokud jsou data poslána na server v čisté textové podobě může je potencionální útočník získat a vystupovat pod zcizenou identitou. Spoléhat se na to, že pravděpodobnost takového činu je malá, není příliš bezpečné. Největší problém metody je tedy v tom, jak autentizační data dopravit k serveru, aby je nikdo po cestě nezískal a aby nebylo možné provést reply atack, kdy útočník ze své adresy pošle celý autentizační paket odchycený z předešlé komunikace.

Jednou možností je mezi klientem a serverem šifrovaný přenos pomocí SSL za využití knihovny OpenSSL. Toto řešení má také výhodu v tom, že lze šifrované spojení následně využívat pro zabezpečení přenášených dat.

Další možností je využití již existujících mechanizmů jako je například protokol Kerberos nebo vytvoření vlastního protokolu vycházejícího z existujících principů, ale upraveného na míru potřebám konkrétní aplikace.

4.1.2 Šifrování

Šifrování přenosu je poměrně náročná operace i s využitím již optimalizované knihovny OpenSSL. Alternativou k opravdovému šifrování může být výpočet XOR funkce z bloku výstupních dat a domluveného klíče. Pokud bude klíč dohodnut při každém navázání komunikace jedná se o relativně přijatelné řešení s ohledem na bezpečnost a rychlost.

4.2 Formát videa

U přenášeného videa je otázkou v jakém formátu data po lince posílat. Na jedné straně stojí jednoduché kódování/dekódování na straně druhé efektivní využití přenosové linky. Je nutné zvážit jaké zdroje jsou k dispozici.

Výhodné je, aby byl server schopen podporovat více formátů podle individuálních potřeb klientů. Mezi základní patří tzv. RAW obraz v odstínech šedi a RGB obraz. V tomto případě se přenáší/ukládají přímo hodnoty jednotlivých pixelů tak jak jsou v obraze. Jedná se o formáty používané při zpracovávání vstupních video dat a režie spojená s jejich vytvořením a zasláním klientům je proto minimální.

V oblasti kamerové techniky je populárním formátem také YUV. Jedná se o barevní model používající k popisu barvy tříprvkový vektor (Y,U,V), kde Y je složka udávající jas a U (Cb) spolu s V (Cr) jsou barevné složky. YUV se například používá u analogového video formátu PAL.

Existuje mnoho variant YUV formátu. Formáty YUV se dělí do dvou skupin - packed a planar, lišící se uložením jasové a barevné složky v paměti. Formáty packed mají uloženy všechny složky YUV do tzv. makropixelů (shluk několika pixelů, např. 4) a jdou po sobě, planar formáty mají uloženy všechny složky zvlášť, tvoří tedy tři virtuální plochy, které jsou ve výsledku složeny dohromady. Dále se pro YUV formáty vžilo tříčíselné označení, např. YUV 4:2:2. Udává vždy poměr mezi počtem barevné složky vůči jasové a někdy i počet bytů na makropixel (někdy se to ale nedodržuje). V tomto případě je poměr 4:2 a barevná složka tedy obsahuje polovinu bodů vůči jasové - na dva jasové body odpovídá pouze jeden barevný. Podobně YUV 4:1:1 obsahuje pouze čtvrtinu barevné složky oproti jasové a YUV 4:4:4 má rovnocenné kódování jasové i barevných složek - měla by tedy být kvalitnější, jenže se přepočítává z YUV 4:2:2 a žádná informace navíc zde tedy není (výhodné pouze pro zpracování, tam se ale používá RGB32) [19].

V praxi nejpoužívanějšími jsou packed formáty s kódováním YUV 4:2:2 z nich konkrétně YUY2 a UYVY, které obsahují stejné složky jen jinak uspořádané (2xY, 1xU, 1xV). Viz Obrázek 3.

Y0 V0 Y1 U0 Y2 V2 Y3 U2 Y4 V4 Y5 U4

Obrázek 3: Příklad uložení formátu YUYV

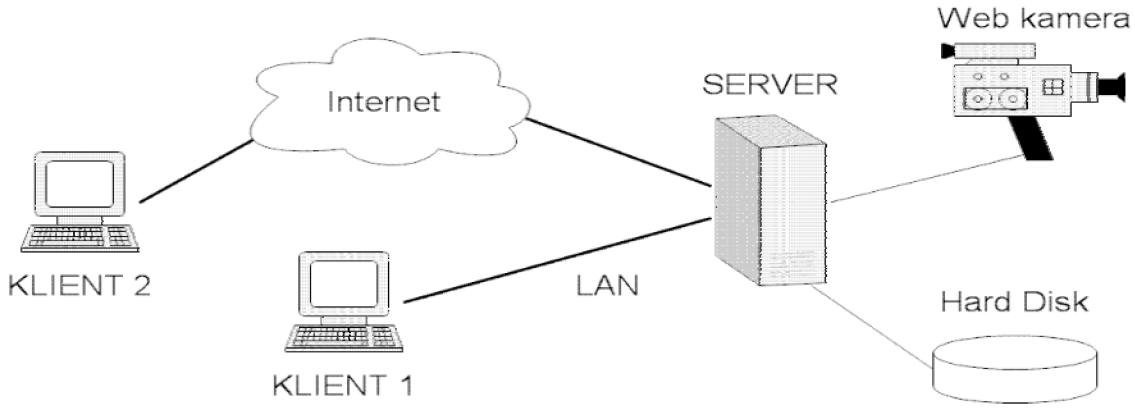
Chybějící hodnoty barevných složek jsou dopočítány interpolací sousedních hodnot. Např. by bylo možné získat V1 jako V1 = (V0 + V2) /2.

Otázkou je jaké další formáty pro kompresi by byly použitelné. Většina řešení pracujících v dnešní době se streamingem videa na internetu jsou uzavřené a licencované programy. Pro open source řešení by bylo možné open source projektu $(LS)^3$ - Libre Streaming, Libre Software, Libre Standards v rámci něhož je implementován streaming idea ve formátu MPEG4 dle standardů RFC3016 a RFC3640. Dále se pracuje na podpoře formátu OGG.

5 Návrh a implementace

Pro implementaci byl zvolen jazyk C++. Je tomu tak především z důvodu širší nabídky standardních funkcí, ale i kvůli možnosti vytvoření objektového návrhu pomocí diagramu tříd v jazyce UML. Kvalitní návrh je schopen odhalit chyby, které by mohly při implementaci jinak vznikat a také přináší přehlednost do celého programu. Díky tomu je snazší určit podstatné závislosti a vytvořit efektivnější a udržovatelný model systému.

Projekt je tvořen ze dvou hlavních částí a to programu *mdd* (Motion detection daemon), který odpovídá serveru na Obrázku 4 a *mdc* (Motion detection client), který slouží pro sledování videa v reálném čase tak, jak je zachycován kamerou. Program mdc je nainstalován na klientech.



Obrázek 4: Celkový pohled na systém

5.1 Server

Tímto termínem je označován program běžící na počítači s připojenou web kamerou. Jsou na něj kladeny následující požadavky:

- snímat obraz z V4L a V4L2
- streaming videa na libovolný počet klientů na lokální síti i po internetu
- detekce pohybu v obraze
- běžet v pozadí (jako systémová služba)
- snadná výměna / tvorba modulů pro detekci
- možnost ukládat obraz ve formě video souboru nebo obrázků na lokální pevný disk
- provádět autentizaci klientů a šifrování dat
- konfigurovatelný přes soubor s nastaveními
- vkládání textu do obrazu (datum, text z externího souboru)
- nastavit čas, kdy je detekce aktivní
- upozornit operátora na pohyb např. e-mailem a přílohou

5.1.1 Konfigurace programu

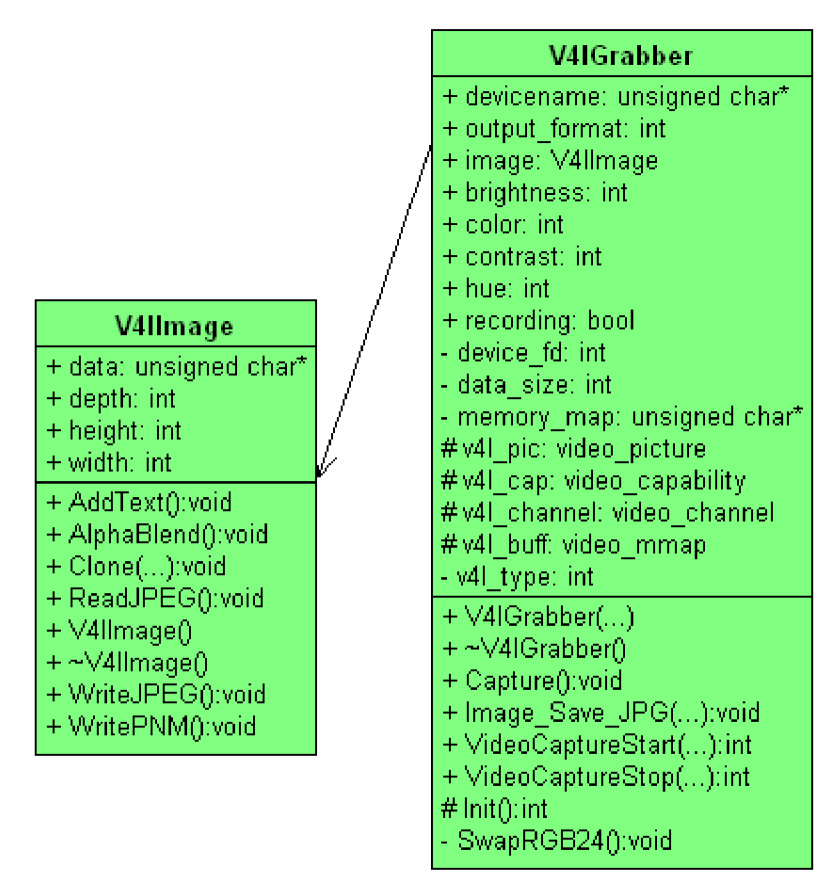
Nastavení programu se s provádí pomocí úprav souboru mdd.conf. Formát parametrů je, jako u většiny konfiguračních souborů v Linuxu. Parametry jsou rozdělovány do sekcí, data vlastní data jsou ve tvaru klíč = hodnota a # značí poznámku.

Pro práci s konfiguračními soubory se využívá staticky přilinkovaná knihovna C++ Config File Library z http://rudeserver.com/config/install.html. Knihovna umožňuje zápis a čtení základních datových typů a není tak třeba explicitně provádět konverze z datového typu na text a zpět.

Popis obsahu konfiguračního souboru a možná nastavení jsou uvedena v Příloze č.1 Manuál.

5.1.2 Snímání obrazu z V4L a V4L2 zařízení

Aby byla implementace pro další využití co nejpřívětivější je vhodné vytvořit několik tříd které budou rozdíly v obou implementacích skrývat a dále poskytovat jednotné rozhraní pro práci se zachycenými video daty. První třídou je *V4lGrabber*, druhou *V4lImage*. *V4lGrabber* slouží ke komunikaci s kamerou pomocí systémových volání. *V4lImage* je pomocnou třídou pro uložení informací o obraze.



Obrázek 5: Diagram tříd V4lGrabber a V4lImage

Pokud je cílem snímat obraz jak z původních V4L zařízení tak i novějších V4L2 kompatibilních, je prvním krokem zjistit jaké zařízení je právě připojeno. Takovou informaci lze získat jednoduchým kódem.

Příklad 8: Detekce verze API

```
if(ioctl(fd, VIDIOGCAP, &cap_V4L) !=-1)
{
       //V4L
       v4l_type = 1;
}
else if (ioctl(fd, VIDIOC_QUERYCAP, &cap_V4L2) !=-1)
{
       //V4L2
       v4l_type = 2;
}
else
{
       // ani V4L ani V4L2
       exit(1);
}
```

Data z kamery budou získávána pomocí namapování obrazu z kamery do paměti alokované ve třídě *V4lGrabber* odkazované ukazatelem memory_map. Obraz který z ovladače kamery přichází je ve formátu BGR24 z toho plyne, že pro jeden pixel budou potřeba 3 byty. Jako základní typ je zvolen unsigned char, nabývající požadovaných hodnot 0-255.

V paměti memory_map se data dynamicky mění podle toho jak ovladač kamery právě nahraje další data. Proto obsahuje třída *V4lGrabber* ukazatel na třídu *V4lImage*, do které se voláním metody capture() zkopírují data z memory_map. Dále pak ke zpracování slouží pouze obraz v paměti odkazované ukazovatelem data ze třídy *V4lImage*. Toto řešení je nutné především v případě, kdy chceme něco do obrazu před jeho uložením přidat. Pokud by se zapisovalo přímo proměnné memory_map mohlo by asynchronní obnovení dat veškeré změny zrušit.

5.1.3 Detekce pohybu

Dalším úkolem serverové aplikace je detekce pohybu v obraze. Jelikož je jedním z cílů vytvořit program v němž bude snadné měnit použitý detekční algoritmus, je vhodné využít dynamicky nahrávaných knihoven. K tomu lze využít funkce popsané v hlavičkovém souboru dlfcn.h. Pomocí funkcí z této knihovny je možné za běhu programu nahrávat do paměti dynamické knihovny (.so) a volat z nich funkce, které známe jménem.

Aby server mohl s pracovat s jednotlivými pluginy (zásuvnými moduly), musí každý z nich implementovat standardní rozhraní. Pojmem rozhraní je zde myšleno sada funkcí pro inicializaci modulu, provádění detekce, ukončení činnosti a uvolnění zdrojů.

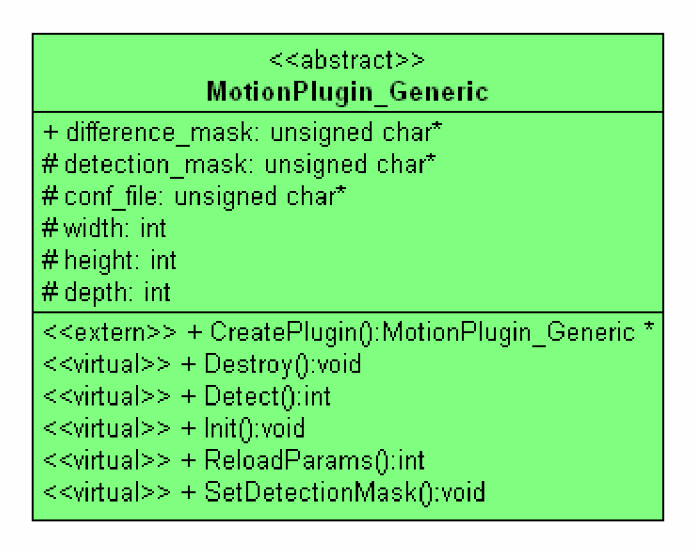
Jako nejvhodnější se jeví implementace pomocí dědičnosti v C++, kdy je definována abstraktní rodičovská třída s virtuálními metodami, která potomkům určí, které metody musí implementovat.

Server pak volá všechny rozšiřující moduly jako třídu *MotionPlugin_Generic* a nezajímá se o implementační detaily jednotlivých modulů.

Jelikož funkce z dlfcn.h je možné použít i v jazyce C lze přirovnat metodu CreatePlugin ke konstruktoru třídy z C++. Každá implementace modulu má pak svoji vlastní.

Někdy může vzniknout potřeba zobrazit, které body nebo větší segmenty obrazu se změnily na to slouží maska rozdílů, rozměry má stejné jako obrázek, ale jednotlivé body nabývají pouze hodnot nula - nezměnil se a jedna - detekována změna. Na úrovni aplikace je následně možné sloučit oba obrázky, případně provést tzv. apha blending a poskytnout tak uživateli přesnější informaci o tom, co se změnilo.

Dále lze definovat masku pozadí. Jedná se o vstupní proměnnou algoritmu, která jinak svými parametry odpovídá masce rozdílu. Nula zde vyjadřuje - v tomto bodě nedetekovat změny, jedna - v tomto bodě provádět detekci.



Obrázek 6: Diagram třídy předka detekce pohybu

Jediným povinným atributem třídy je conf_file s relativní cestou ke konfiguračnímu souboru rozšiřujícího modulu. V tomto souboru se pak nachází konkrétní parametry potřebné pro běh algoritmu. Pokud se konfigurační soubor změní lze parametry obnovit pomocí funkce ReloadParams(). Je doporučené, aby se konfigurační soubor jmenoval jako soubor modulu s příponou .conf.

Metoda Detect() má tři návratové hodnoty. Jsou to:

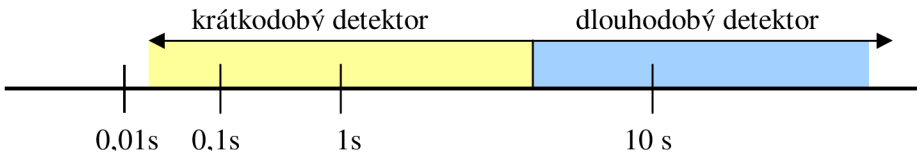
- DETECT_NONE - vrácena pokud není detekován žádný pohyb
- DETECT_MOTION - pokud je detekován potenciální pohyb, slouží ke spuštění nahrávání, snímání obrázku
- DETECT_ALARM - v případě že je potvrzen, pohyb a má se zaslat upozornění

Pluginy pro detekci pohybu by měly být umístěné v podadresáři motion v hlavním adresáři programu.

Moduly mohou v principu fungovat ve dvou různých módech. Buď jako krátkodobý detektor nebo jako dlouhodobý detektor.

Krátkodobý detektor detekuje rychle probíhající děje povětšinou změnu mezi dvěma sousedními snímky videa nebo snímky s krátkým mezi intervalem. Interval nesmí být příliš krátký, neboť pak by snímky neobsahovaly dostatek rozdílů a detekce pohybu by nebyla úspěšná. Pomalu pohybující se zloděj nebo jinak nebezpečná osoba by se tedy mohla nepozorovaně před detektorem proplížit. Interval by se měl pohybovat podle rozměru scény od desetin sekundy do několika sekund.

Dlouhodobý detektor pracuje se snímky s větším časovým rozestupem. Je tedy schopný detekovat pomalu se pohybující předměty, ale rychlý pohyb mu může uniknout. Zde je vhodný interval od několika sekund do desítek sekund.



Obrázek 7: Interval krátkodobého a dlouhodobého detektoru

5.1.3.1 Modul detekce rozdílných pixelů

Soubor modulu: mdp_simple.so

Konfigurační soubor: mdp_simple.conf

První a nejméně sofistikovaným algoritmem je výpočet počtu rozdílných bodů popsaný rovnicí č. 2. Tento algoritmus má dva prahové parametry. Prvním je hodnota T_1 o kterou se má bod na stejné pozici lišit, aby byl považován za změněný viz. rovnice 6. Před samotným výpočtem se barva aktuálních bodů převede do odstínů šedi dle rovnice 1.

$$f(x,y) = \left| p_t[x,y] - p_{t-1}[x,y] \right| \begin{cases} f(x,y) \geq T1 \Rightarrow dif = 0 \\ f(x,y) < T1 \Rightarrow dif = 1 \end{cases}$$

Rovnice 6: Rozdílnost bodů

$p_t[x,y]$ je bod na souřadnicích x,y v aktuálním obrázku, $p_{t-1}[x,y]$ bod na souřadnicích x,y v předchozím obrázku.

Druhým je počet bodů T_2, které se musí změnit, aby byl v obrazu detekován pohyb - rovnice 7. Tuto hodnotu je vhodné uvádět v procentech neboť se bude lišit podle rozlišení vstupního obrazu.

$$\sum_{i=0}^{n} dif_i > T_2(\%)$$

Rovnice 7: Podmínka detekce pohybu

Detekce založená na pouhém počtu rozdílných bodů obrazu je lehce implementovatelná a není příliš náročná na systémové zdroje. Pro fungování stačí pouze jeden alokovaný prostor s daty z předešlého běhu. Nevýhodou je, že neposkytuje informaci ucelenější informaci o tom co se v obraze změnilo a při špatném nastavení je i poměrně náchylná na šum.

5.1.3.2 Detekce rozdílných segmentů

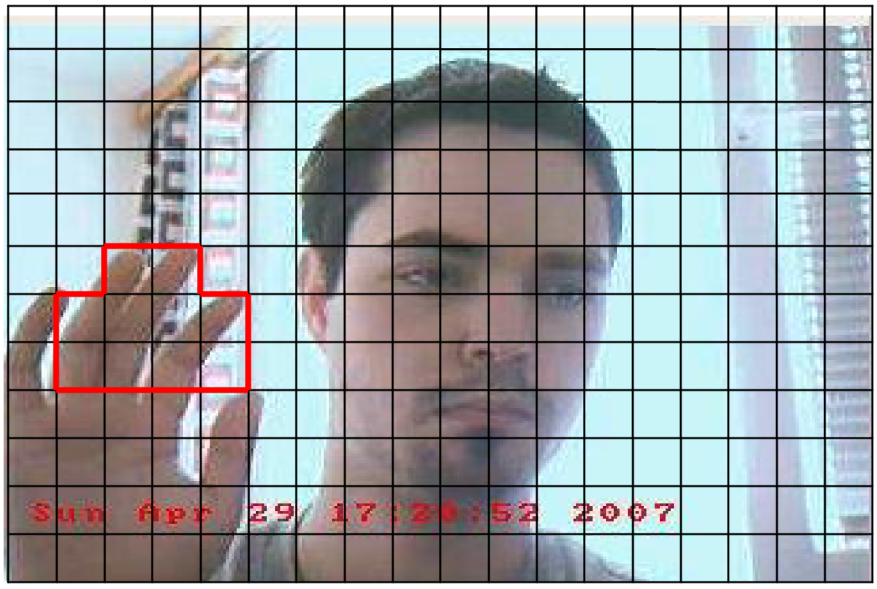
Soubor modulu: mdp_segment.so

Konfigurační soubor: mdp_segment.conf

Metoda detekce rozdílných segmentů přináší vyšší úroveň abstrakce do detekce pohybu.Obraz je pro tento algoritmus rozdělen pravoúhlou mřížkou na čtvercové části o specifikované velikosti. V rámci každého segmentu se provádí detekce počtu rozdílných bodů, jako by se jednalo o samostatný obrázek tzn. dle rovnic 6 a 7. Na obrázku č. 8 je příklad rozdělení obrazu do segmentů a červenou čarou je ohraničena oblast ve které byl detekován pohyb (tomto případě prstů). Ohraničení změněných segmentů je předáváno v masce rozdílů. Parametry (jaký bude výstup v masce) se nastavují) v mdp_segment.conf.

Tento algoritmus potřebuje navíc paměťové pole uchovávající informaci o tom, zda se daný segment změnil či nikoliv. Parametr T_1 zůstává stejný, T_2 udává počet segmentů, které se musí změnit aby byl detekován pohyb.

Výhodou této metody je, že lze sledovat pohyb segmentu nebo oblasti složené z několika segmentů. V případě že by se připojená kamera dokázala otáčet podle příkazů z počítače, lze pohybující se objekt sledovat.



Obrázek 8: Detekce rozdílných segmentů

5.1.3.3 Detekce pozadí pomocí gausiánů

Soubor modulu: mdp_1gausian.so

Konfigurační soubor: mdp_1gausian.conf

Jedná se o nejkvalitnější ze zde uváděných algoritmů. Vytváří statistický model popisující pozadí pomocí rovnic gausiánů. Slovem rovnic jsou míněny konkrétní parametry pro výpočet rovnice. V praxi používané kvalitní modely využívají více rovnic pro popis jednoho bodu. S tím však narůstá časová i paměťová složitost implementace.

Algoritmus využívá dynamický model pozadí. Díky tomu je možné reagovat na události jako je změna intenzity osvětlení. To je pro statický model pozadí, který je vypočítán pro scénu jen jednou při inicializaci, neřešitelný problém.

5.1.4 Záznam obrazu

Nahrávání video sekvencí a ukládání obrázků je dalším požadavkem na systém. Otázkou je jak data ukládat s ohledem na snadné přehrání, kompresi a přenositelnost. Vymýšlet nové formáty by bylo v dané situaci nevhodné a neefektivní, proto bylo v projektu využito již hotových knihoven. Pro kompresi obrázků libjpeg a FFmpeg pro práci s videem.

Třída *V4lGrabber* má na starost zpracování a ukládání obrazu dat. Jako formát statických obrázků byl zvolen JPEG pro video MPEG2 komprese. MPEG2 neposkytuje příliš vysokou úroveň komprese ale na druhou stranu se jedná o velmi rozšířený formát.

5.1.5 Přenos obrazu

Přenos dat ke klientům je v této verzi řešen pomocí zasílání obrázků ve formátu JPEG. Nejedná se tedy o streaming v pravém sova smyslu, který bývá řešen pomocí video formátu, ale spíše o přenos souborů za využití TCP. Tento stav není definitivní. Protokol umožňuje vyjednat i jiné formáty přenosu a proto je možné následně doprogramovat i jiné metody.

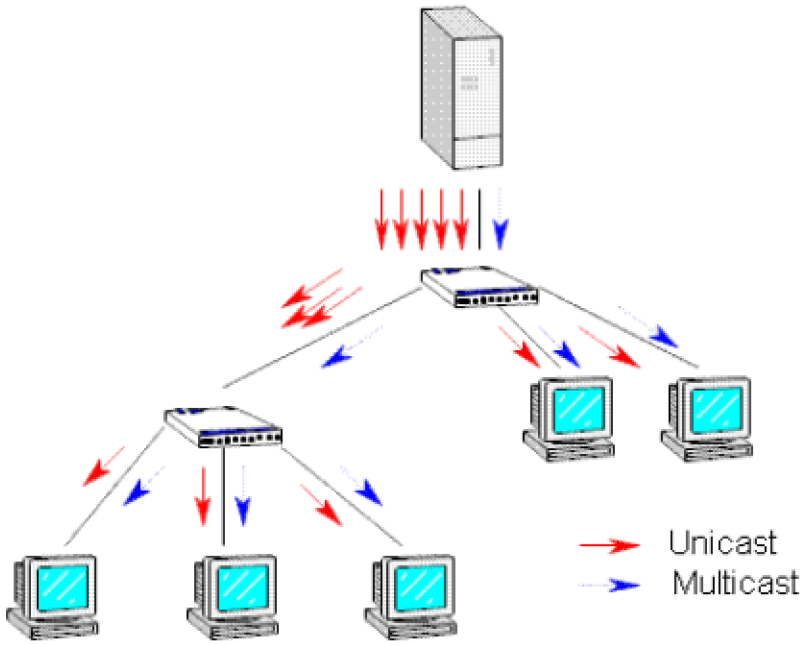
Použitím TCP je možné získat spolehlivý přenosový kanál a případně použít knihovny OpenSSL pro vytvoření šifrovaného kanálu. Jelikož mezi požadavky je možnost připojení "libovolného" počtu klientů, musí k tomu být server přizpůsoben. To by znamenalo využít TCP socketů v neblokujícím módu a funkce select.

Přes všechny uvedené klady má protokol TCP nevýhodu ve větší zátěži sítě. S přihlédnutím k podmínce na libovolný počet klientů není příliš vhodné vybrat pro transport obrazu protokol TCP, protože může síť nadměrně přetěžovat. V praxi se také prakticky nesetkáme se streamingem v

reálném čase pomocí TCP. S ohledem na nároky na výpočetní výkon nebude použito ani SSL spojení a tak není TCP potřeba.

Nejméně náročným způsobem přenosu stejných dat více klientům je TCP broadcast nebo multicast. Broadcast je nevhodný, protože se šíří pouze v rámci lokální sítě. Multicast je možné šířit i přes hranice lokální sítě, pokud to směrovače podporují. Důvod pro využití multicastu je patrný z obrázku 9. Je patrné, že červených šipek, označujících unicastové pakety, se v síti vyskytuje mnohem víc. Problémem je hlavně přetížení serveru a linek v jeho blízkosti.

Nevýhodou multicastu je, že směrovače na cestě ke klientům musí podporovat šíření paketů. V prostředí internetu poskytovatelé připojení většinou rekonfiguraci svých strojů nedovolí. Pak není možné multicast používat. Z toho důvodu jsou implementovány oba typy přenosu. V konfiguračním souboru je možné nastavit, který typ přenosu se použije. Detaily nastavení jsou k nalezení v příloze č. 1 Manuál.



Obrázek 9: Rozdíl unicastu a multicastu

5.2 Protokol aplikační vrstvy

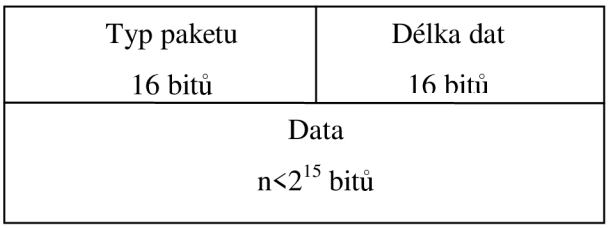
Pro přenos dat mezi dvěma body na sítí je nutné data převést do takového formátu, aby mu obě strany rozuměly a příjemce byl schopný co nejlépe zrekonstruovat informaci kterou odesilatel do sítě vyslal.

Tak jako je na úrovni síťové vrstvy použit IP protokol a na transportní TCP, musí být na určitý i na aplikační úrovni.

Na protokol jsou kladeny následující požadavky:

- klient požádá o zasílání video dat
- server má možnost požádat o autentizaci jménem a sdíleným heslem
- server zašle podporované formáty dat
- klient vybere formát a požádá o data
- klient posílá udržovací paket
- klient ukončí komunikaci
- server ukončí zasílání pokud nedostane po dva intervaly udržovací paket
- pokud jedna strana obdrží typ paketu, který nečeká, vyžádá obnovení spojení
- server neopakuje odesílání dat, klient se s tím dokáže vypořádat

Navrhovaný protokol slouží pro streaming videa z webové kamery, proto bude dále nazýván jako *Web Cam Streaming Protocol - WCSP*. Protokol komunikuje pomocí zpráv v binární formátu. Strukturu paketu ukazuje obrázek 10.

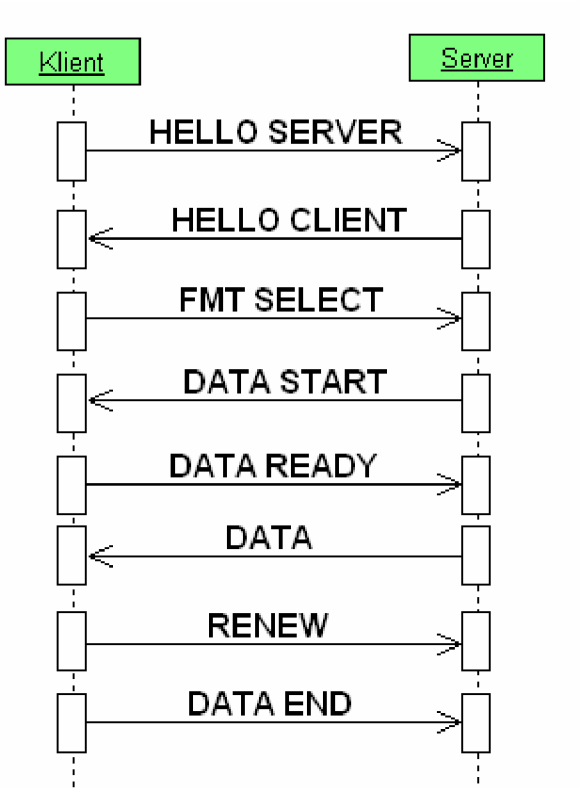


Obrázek 10: Standardní hlavička

Prvním polem paketu je jeho *Typ*. Podle typu paketu je zpracováváno pole *Data*, neboť u každého typu obsahuje jiná data.

Druhým polem je 16ti bitová *Délka dat*. Protože je požadováno, aby se přenesl celý paket aplikační vrstvy musí mít velikost takovou aby se vešel do TCP paketu. Tato nutnost plyne především z přenášení datových paketů, u kterých nutně potřebujeme hlavičku s identifikací do jakého snímku a na jaký offset data patří.

Posloupnost zpráv vyměněných mezi serverem a klientem v rámci základní komunikace je vyobrazena na obrázku 11.



Obrázek 11: Základní komunikace pomocí protokolu WCSP - bez chyb

Pokud jedna ze stran dostane paket, který by v posloupnosti nečekala odpoví zprávou ERROR s chybou 0x01. Celou komunikaci je poté nutné inicializovat znova.

5.2.1 Popis zpráv

HELLO SERVER - zpráva zahajující komunikaci. Klient v datové části posílá serveru svoji *identifikaci* v textové podobě. Může být zaslána hodnota *anonymous*.

Typ paketu (16 b)	Délka dat (16 b)
Identifikace | "anonymous"(n * b)	

Obrázek 12: Formát zprávy HELLO SERVER

V takovém případě server nepošle výzvu k autentizaci a pokud ji potřebuje vrátí zprávu *ERROR* s číslem chyby 0x02.

HELLO CLIENT - server odpovídá na výzvu. V odpovědi posílí výčet podporovaných formátů. *Formáty* jsou uvedeny za sebou jako 32bitové konstanty.

Typ paketu (16 b)	Délka dat (16 b)
Šířka obrázku (16 b)	Výška obrázku (16 b)
Počet formátů (32 b)	
Formáty (n * 32 b)	

Obrázek 13: Formát zprávy HELLO CLIENT

Navrhované formáty pro verzi *WCSP v.1*:

RAW_8 - 0x01 - nekomprimovaný obrázek s 8bity na pixel

RAW_24 - 0x02 - nekomprimovaný obrázek s 24bity na pixel

UYVY - 0x04 - nekomprimovaný obrázek v rozšířeném formátu videa

JPEG - 0x10 - obrázek v komprimovaném formátu JPEG

FMT SELECT - Klient odešle v odpovědi formát o který má zájem. Pokud si žádný formát z nabízených nevybere vrátí zprávu *ERROR* s chybou 0x03.

Typ paketu (16 b)	Délka dat (16 b)
Formát (32 b)	

Obrázek 14: Formát zprávy FMT SELECT

DATA START - Server nastaví výstupní formát dat na požadovaný a oznámí klientovy, že je schopen vysílat data. *Čas obnovy* je buď zadán staticky nebo se spočítá z tzv. Round Trip Time předcházejících paketů. Po uplynutí tohoto intervalu server přestane klientovy posílat datagramy, pokud od něho neobdrží zprávu *RENEW*

Typ paketu (16 b)	Délka dat (16 b)
Čas obnovy v ns (32 b)	

Obrázek 14: Formát zprávy DATA START

DATA READY - po tom co se připraví na příjem dat (provede alokaci vyrovnávacích paměti..), oznámí že je připraven.

DATA - zpráva obsahující obrazová data. *Sekvenční číslo* obrázku se pro první snímek volí náhodně a pro každý další snímek se zvětší o jedna. Pokud se dostane na maximální hodnotu rozsahu změní se v dalším kroku na minimální hodnotu. *Offset* udává na jakou pozici ve výstupním obrázku data umístit. Tento parametr je zde především kvůli případnému použití protokolu TCP. *Velikost celého obrázku* je číslo udávající velikost v bytech.

Na programátorovi je, jak se vypořádá s chybějícími daty při zobrazování. Doporučuje se vyplnit takovou plochu jednou barvou, aby uživatel poznal že se jedná o chybu v přenosu.

Typ paketu (16 b)	Délka dat (16 b)
Sekvenční číslo obrázku (32 b)	
Offset dat (32 b)	
Velikost celého obrázku (32 b)	
Data (n * x b)	

Obrázek 15: Formát zprávy DATA

RENEW - oznámení, kterým klient potvrzuje, že chce nadále dostávat od video serveru data. Po uplynutí doby časového limitu pošle server klientovi *DATA END* zprávu.

DATA END - v případě že klient chce spojení ukončit pošle tuto zprávu a server ho vyjme ze seznamu příjemců. Dále viz. *RENEW*

SERVER STOP - pokud z nějakého důvodu končí server svoji činnost (a není to násilným způsobem) oznámí to touto zprávou příjemcům.

ERROR - Zpráva o chybě v komunikaci. Příjemce i odesílatel se nastaví do výchozího stavu.

Typ paketu (16 b)	Délka dat (16 b)
Číslo chyby (32 b)	
Textový popis chyby (32 b)	

Obrázek 16: Formát zprávy ERROR

Výčet chyb ve WCSP v.1:

0x01 - Bad sequence - odesilatel obdržel zprávu v neočekávaném pořadí.

0x02 - Authentication required - je vyžadována autentizace klientů.

0x03 - Can not choose - klient si nevybral z nabízených formátů.

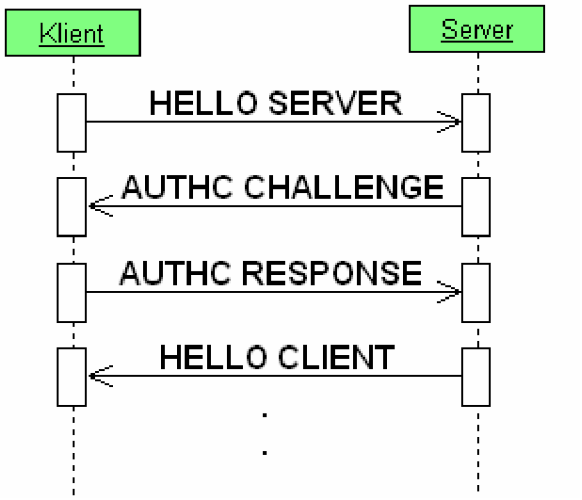
0x04 - Client authentication failed - autentizace klienta se nezdařila

0x05 - Encryption required - server vyžaduje šifrování spojení

0x06 - Encryption method refused - server neakceptuje vybranou metodu

5.2.1.1 Autentizace

Prvním rozšířením je možnost autentizace klienta serveru. Tak je zabráněno stavu kdy by data z kamery mohl sledovat někdo nepovolaný.



Obrázek 17: Rozšíření o autentizaci

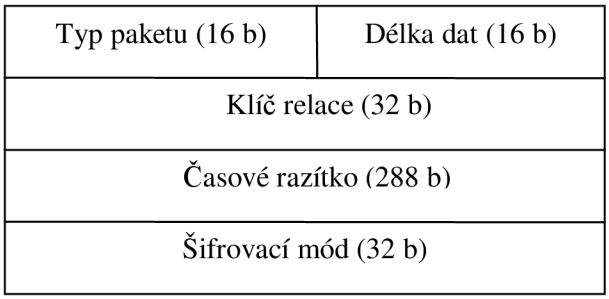
AUTHC CHALLENGE - v této zprávě posílá server klientovi zprávu, kterou chce od klienta ověřit totožnost. Zpráva obsahuje datum a 32bitové číslo sloužící jako klíč relace. Klíč relace není pro klienta důležitý, pokud si nepřeje navázat šifrované spojení.

Typ paketu (16 b)	Délka dat (16 b)
Klíč relace (32 b)	
Časové razítko (288 b)	

Obrázek 18: Formát zprávy AUTHC CHALLENGE

AUTHC RESPONSE - Klient odpovídá aktualizovaným datem a inkrementovanou hodnotou klíče relace. V poli šifrovací mód je standardně hodnota 0x00 značící že šifrování není požadováno. Více o šifrování v oddíle 5.2.1.1 šifrování.

Pokud server obdrží špatnou hodnotu klíče relace, odpoví zprávou *ERROR* s číslem 0x04. V případě že server vyžaduje šifrování vrátí *ERROR* s hodnotou 0x05.



Obrázek 19: Formát zprávy AUTHC RESPONSE

Otázkou je jak provést autentizaci, tak aby útočník nemohl snadno odposlechnout autorizační pakety, sestavit slovník, za předpokladu že nezná algoritmus, a pak se vydávat za právoplatného příjemce vysílání. Řešením problému je využití kryptografie. Nejjednodušším postupem pro zašifrování dat je symetrická kryptografie se sdíleným klíčem. V tomto případě je na serveru pro spojení s určitým klientem definován jeden klíč. Shodný klíč je nutné umístit i do konfiguračního souboru klienta. Pro jednoduchost je uvažováno, že k tomuto souboru budou mít přístup pouze oprávnění uživatelé a proto není třeba heslo chránit a může být uloženo v plain textové podobě.

Navrhovaný postup zabezpečení není v žádném případě neproniknutelný, nabízí však možnost alespoň základního utajení. Nepřenáší se heslo resp. soukromý klíč, proto není vystaven riziku přímého napadení. Datum v autentizační zprávě zabraňuje jednoduchému útoku pomocí slovníku zachycených zpráv.

Autentizační paket bude šifrován symetrickou šifrou DES. Jedná se o standard v kryptografii. Přestože dnes již není považován za zcela bezpečný pro tento projekt je postačující. Nepředpokládá se, že by někdo útočil na tento systém za využití stovek či tisíců počítačů.

5.2.1.2 Šifrování

Další rozšířením je možnost utajit přenášená data. Úmyslně nebylo použito slova šifrovat, protože bude implementována velmi primitivní metoda. Metoda je pracovně nazvána XOR32. Přenášená data jsou pozměněna pomocí operace XOR a klíče relace z fáze autentizace.

CIPHER OK - pokud serve souhlasí s použitím vybrané metody zabezpečení vrátí tuto zprávu. Jinak je vrácena *ERROR* s hodnotou 0x06.

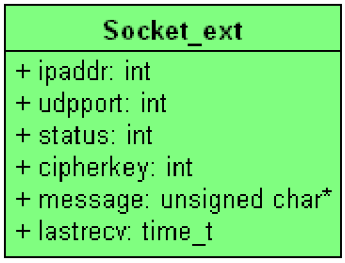
Výčet šifrování ve *WCSP v.1*:

0x00 - No cipher - nešifrovat

0x01 - XOR32 - šifrování pomocí operace XOR a 32bitového klíče

5.2.2 Implementace protokolu na straně serveru

Server je schopen obsluhovat téměř libovolný počet klientů. Každý klient se nachází v nějakém stavu vyjednávání komunikace a očekává data v určitém formátu. Pro zjednodušení se předpokládá, že po tom, kdy se klient dostane na řadu, se mu odešle celý obrázek ve specifikovaném formátu. Informace o jednotlivých spojeních jsou uloženy ve třídě Socket_ext.



Obrázek 19: Třída Socket_ext

Atribut *ipaddr* obsahuje ip adresu a *udpport* port na klientovi. *Status* značí v jakém stavu se nachází komunikace s klientem. Pod atributem *cipherkey* je uloženo číslo k autentizaci a případný klíč relace. *Message* je určen pro pomocná data ke spojení, která by server mohl potřebovat uložit. Aby bylo možné určit kdy se má spojení ukončit je zde atribut *lastrecv*. V tom je uloženo kdy naposled klient potvrdil příjem dat. Pokud hodnota přesáhne stanovený práh, odstraní se tato instance třídy ze seznamu pro odesílání.

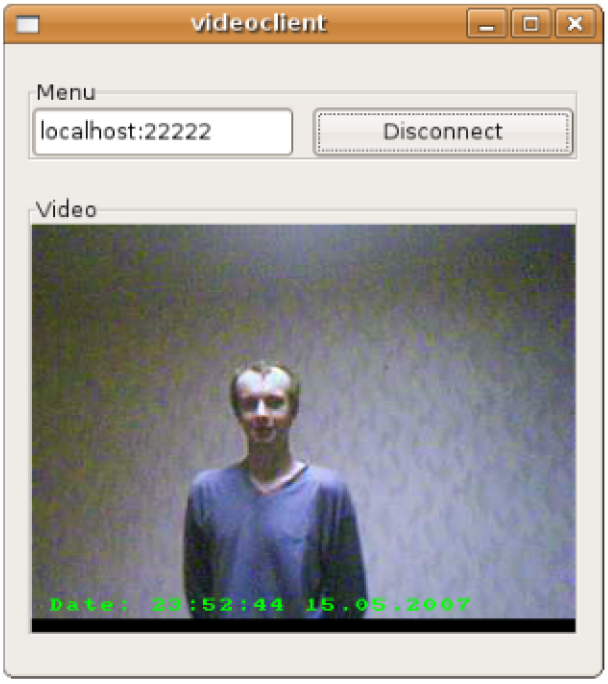
5.3 Klient

Klientskou část projektu tvoří program pro příjem videa. Tímto termínem je označována aplikace běžící na počítači uživatele, který chce sledovat obraz kamery. Jsou na něho kladeny následující požadavky:

- zadat adresu a port serveru
- autentizace serveru heslem
- zobrazování videa

Tato část projektu se skládá z uživatelského rozhraní pro zobrazení videa a síťového klienta. Jedná se o velmi jednoduchou aplikaci, která nemá žádné další funkce.

Pro tvorbu uživatelského rozhraní využívá knihovnu gtkmm. Okno aplikace obsahuje pole pro zadání adresy tlačítko pro připojení/odpojení k serveru a plochu pro zobrazení obrazu viz obrázek 20. Pokud server požaduje autentizaci otevře klient nové okno s poli pro vyplnění loginu a hesla.



Obrázek 20: Sledování scény

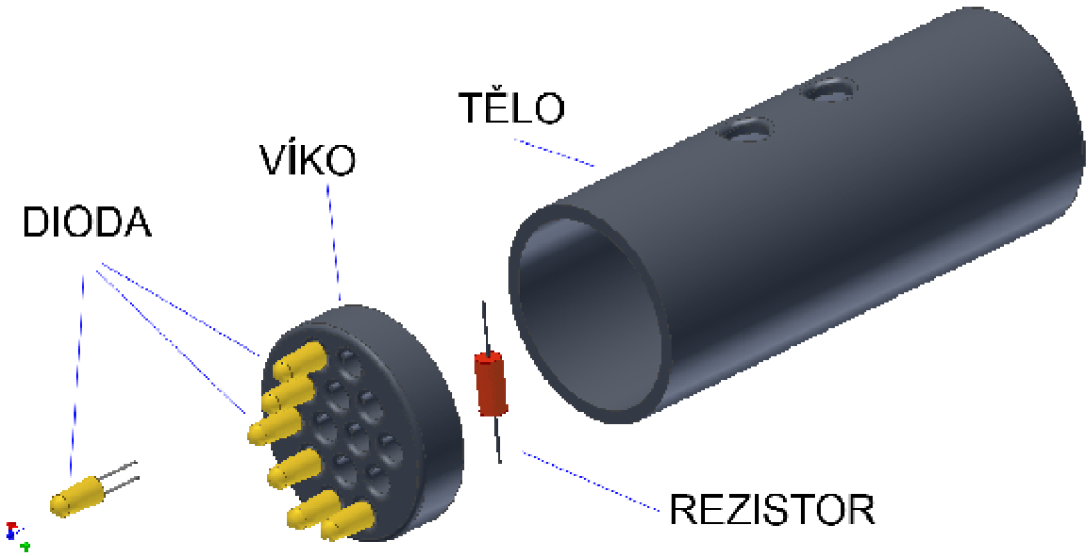
Zda se v obraze objeví datum, přednastavený text nebo zvýraznění změny v obraze závisí pouze na nastavení serveru a klient je v této verzi nemůže nijak ovlivnit.

5.4 Přisvětlení scény

Pokud intenzita světla před kamerou není z nějakého důvodu dostatečná, je třeba světlo do scény dodat. Nedostatečné osvětlení může být způsobeno například noční dobou, zataženou oblohou ve dne nebo neosvětlenou místností.

5.4.1 Konstrukce svítilny

Pro přisvětlení scény nutný externí zdroj světla. Součástí této práce je i návrh prototypu svítilny složené z LED diod napájené pomocí USB kabelu z PC. Konstrukce svítilny je vyobrazena na obrázku 21. Tělo se skládá z PVC trubky o průměru 4cm a délky 10cm. V těle jsou vyvrtány dva otvory pro montáž úchytného mechanizmu. Konec těla uzavírá víko s otvory pro diody. Z druhé strany je do těla přiveden USB kabel pro napájení.



Obrázek 21: Konstrukce LED osvětlení

5.4.2 Specifikace USB

Uvádět celou specifikaci USB není na tomto místě pravděpodobně nutné. Pro potřeby návrhu se stáčí seznámit se standardy fyzické vrstvy. Dle specifikace nalezené na USB in a NutShell [20] je USB kabel tvořen čtyřmi vodiči a je zakončen konektorem typu A nebo B viz obrázek 22.

Obrázek 22: Typy USB konektorů

Číslo vývodu	Barva izolace kabelu	Funkce
1	Červená	V_{BUS} (5V)
2	Bílá	D-
3	Zelená	D+
4	Černá	Zem

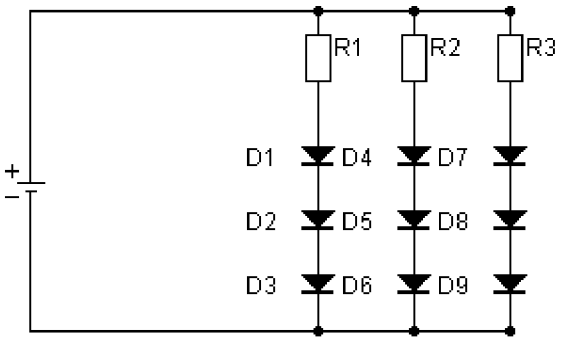
Tabulka 1: Popis vodičů

Vodiče D+ a D- slouží jako diferenciální datová spojení a nebudou v daném případě použity. Důležitý je červený vodič s 5V a černý představující zem.

Druhým významným parametrem je maximální proud v obvodu, který má hodnotu 500mA.

5.4.3 Elektrický obvod

Diody mohou kromě viditelného spektra produkovat i záření v infračerveném spektru (940nm). To není lidské oko schopno zachytit, ale čip kamery ano. Lze tak získat nepozorovatelné osvětlení . Nevýhodou řešení s IR diodami je černobílý obraz. Svítilna samozřejmě může používat i normální diody. Pro úsporu místa i spotřeby je výhodné zapojit více diod do série. Počet diod v sérii je omezen napětím které diody k práci potřebují.



Obrázek 23: Zapojení se třemi LED v sérii

Hodnota rezistoru v jednotlivých větví obvodu na obrázku 23 se vypočítá jako:

$$R_i = \frac{U_{zdroje} - (U_{D1} + U_{D2} + U_{D3})}{I_D}$$

Rovnice 7: Hodnota rezistoru

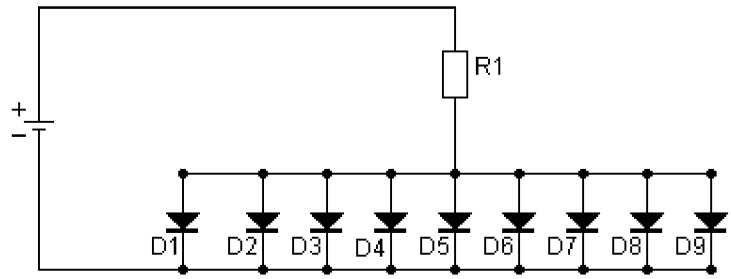
Další možností je zapojení s jedním společným rezistorem. Od něho jsou rozvedeny vodiče k jednotlivým diodám jak je patrné z obrázku 24. Při tomto zapojení je nutné použít výkonový rezistor dimenzovaný na výkon větší než P.

$$P = U_{rozdílové} * I_{celkové}$$

Rovnice 8: Výkon rezistoru

$$R = \frac{U_{celkové} - U_{LED}}{\sum_{k=1}^{n} I_{LED\,k}}$$

Rovnice 9: Hodnota rezistoru



Obrázek 24: Zapojení LED paralelně s jedním rezistorem

6 Závěr

V rámci této práce byl vytvořen kompletní software pro detekci pohybu ve sledované scéně pomocí webové kamery pod OS Linux. Detekovaný pohyb je možné zaznamenat na pevný disk ve formě obrázku nebo videa, případně odeslat jako přílohu emailu na předdefinovanou adresu.

V další části je popis klientské aplikace. Ta souží pro sledování videa vysílaného serverovou aplikací v reálném čase. K tomuto účelu je vytvořen jednoduchý protokol pro přihlášení k serveru a žádost o zasílání dat. Obrazová data jsou přenášena jako samostatné obrázky, ne jako video soubor. V takovém případě není počet snímků za sekundu pevně stanovený ale odvíjí se od zatížení sítě a serveru.

Společenský vývoj klade stále větší důraz na bezpečnost a ochranu majetku a soukromí. Existující aplikace tyto požadavky poněkud ignorují. Většinou nechají na uživateli aby pomocí SSH nebo SSL vytvořil tunel či zabezpečené spojení k požadovaným datům. Navržená implementace přináší variantu řešení bezpečnostních problémů. Je zde navržen a implementován mechanizmus, který umožní přenášená data sledovat jen oprávněnému uživateli tak, že server má možnost vyžádat autentizaci klientů a šifrovat spojení.

Největší výhodou, kterou aplikace přináší je snadné použití dalších algoritmů pro detekci pohybu. Jednotlivé algoritmy jsou k programu připojovány ve formě dynamických knihoven s pevně daným rozhraním. Program tedy není nutné kvůli přidání algoritmu znovu zkompilovat. Stačí jen nahrát modul a změnit konfigurační soubor.

Již nyní je vidět několik oblastí, ve kterých by bylo možné program dále vylepšit. Jedná se především o bezpečnost a kvalitnější detekční algoritmy.

V případě požadavku na vyšší stupeň zabezpečení by bylo vhodné implementovat serverové certifikáty jako je tomu u SSL. Pak nemusí server s klientem sdílet symetrické klíče pro šifrování. Instalace klientů na nové stroje je tím pádem snazší. Za nevýhodu lze považovat zátěž vzniklou použitím asymetrické kryptografie při ustanovování spojení.

Právě zatížení serveru a nároky na systémové zdroje jsou hlavní nevýhodou i druhého navrhovaného rozšíření, kterým jsou kvalitnější detekční algoritmy. Pokud stroj, na němž je server nainstalovaný neslouží pouze k zbezpeční objektu, je nutné toto negativum zvážit.

Literatura

[1] MATHEW, Niel, STONES, Richard. Linux : Začínáme programovat. Praha : Computer Press, 2000. 897 s. ISBN 80-7226-307-2.

[2] Daemon [online]. 2005 [cit. 2006-12-16]. Dostupný z WWW: <http://wiki.linuxquestions.org/wiki/Daemon>.

[3] Wikipedia [online]. [2006] [cit. 2006-12-16]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Main_Page>.

[4] BERAN, Martin. Programování pro X Window System (2) : GTK+ [online]. 2004 [cit. 2006-12-16]. Dostupný z WWW: <http://www.root.cz/clanky/programovani-x-window-system-gtk-plus/>.

[5] Ing. GRYGÁREK Ph.D., Petr . Rozhraní BSD Sockets [online].[cit. 2006-12-17]. Dostupný z WWW: <[I] http://www.cs.vsb.cz/grygarek/PS/sockets.html>.

[6] PETERKA, Jiří. Transportní rozhraní - BSD Sockets [online]. [1993] [cit. 1993-12-17]. Dostupný z WWW: <http://www.earchiv.cz/a93/a315c110.php3>.

[7] POSIX Threads Programming [online]. 2006 [cit. 2006-12-18]. Dostupný z WWW: <http://www.llnl.gov/computing/tutorials/pthreads/>.

[8] MATYS, Jakub. Programování pod Linuxem pro všechny : 18 [online]. 2004 [cit. 2006-12-18]. Dostupný z WWW: <http://www.root.cz/clanky/programovani-pod-linuxem-pro-vsechny-18/>.

[9] Cvičení 3 – Vlákna (threads) [online]. [2006] [cit. 2006-12-18]. Dostupný z WWW: <http://dce.felk.cvut.cz/pos/cv3/index.html>.

[10] MUYS, Andrae. A Pthreads Tutorial [online]. 2006 [cit. 2006-12-18]. Dostupný z WWW: <http://www.cs.nmsu.edu/~jcook/Tools/pthreads/pthreads.html>.

[11] SCHIMEK, Michael H. Video for Linux Two API Specification [online]. 2006 [cit. 2006-12-18]. Dostupný z WWW: <http://v4l2spec.bytesex.org/spec>.

[12] FARKAŠOVÁ, Blanka, KRČÁL, Martin. Projekt Bibliografické citace [online]. c2006 [cit. 2006-12-16]. Dostupný z WWW: < http://www.citace.com/clanky.php>.

[13] Webcam [online]. 2006 [cit. 2006-12-24]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Web_cam>.

[14] Video for Linux resources [online]. 2006 [cit. 2006-12-25]. Dostupný z WWW: <http://www.exploits.org/v4l/>.

[15] KIRILLOV, Andrew. Motion Detection Algorithms [online]. 2006 [cit. 2006-12-24]. Dostupný z WWW: <http://www.codeproject.com/cs/media/Motion_Detection.asp>.

[16] ŠERÝ, Ondřej. Odečítání pozadí a sledování [online]. c2004 [cit. 2006-12-24]. Dostupný z WWW: <http://cgg.ms.mff.cuni.cz/~pepca/i218/tracking.pdf>

[17] GRIMSON, W.E.L., STAUFFER, C.. Adaptive background mixture models for real-time tracking [online]. 1998 [cit. 2006-12-24]. Dostupný z WWW: <http://www.ai.mit.edu/projects/vsam/Publications/stauffer_cvpr98_track.pdf>.

[18] Autentizace a autorizace [online]. 2006 [cit. 2006-12-25]. Dostupný z WWW: <http://www.secunet.cz/services/Technologie/Autentizace.html>.

[19] Formáty obrazu videa [online]. 2001 [cit. 2006-12-28]. Dostupný z WWW: <http://www.tvfreak.cz/modules.php?name=News&file=print&id=277>.

[20] PEACOCK, Craig. *USB in a NutShell* [online]. 2007 [cit. 2007-05-11]. Dostupný z WWW: <http://www.beyondlogic.org/usbnutshell/usb2.htm>.

[21] MICHAL BLÁHA. Hledám betatestery softwaru pro detekci pohybu [online]. 2005 [cit. 2007-04-28]. Dostupný z WWW: <[1] http://blog.vyvojar.cz/michal/archive/2005/05/02/5540.aspx>.

Seznam příloh

Příloha 1. Manuál programu

Instalace

Pro zkompilování a běh programu jsou v systému potřebné následující knihovny:

- libjpeg (v6b) z http://www.ijg.org/
- FFmpeg (v0.4.9) z http://ffmpeg.mplayerhq.hu/download.html
- C++ Config File Library (v5.0.5) z http://rudeserver.com/config/

Zdrojové soubory knihoven jsou také k nalezení na CD v adresáři libraries.

Postup instalace:

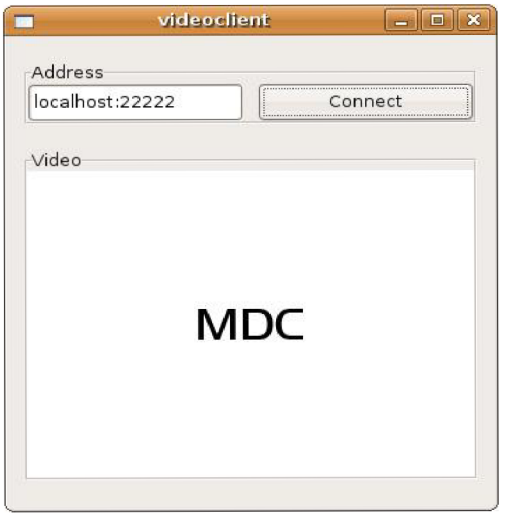
```
1. make
2. make install
```

Tím se vytvoří soubor tři soubory

- /usr/bin/mdc - klientská aplikace
- /usr/bin/mdd - detekční a serverová aplikace
- /usr/share/mdd/mdd.conf - konfigurační soubor
- /usr/share/mdd/auth.conf - soubor pro autentizaci

Klient

Klientská část pro sledování videa se jmenuje mdc (motion detection client). Po spuštění se zobrazí okno aplikace (obrázek 25).



Obrázek 25: Okno klientské aplikace

Připojení k severu

Do pole address se vyplní IP nebo DNS adresa serveru a dvojtečkou oddělený port. Tlačítkem Connect se klient pokusí připojit k serveru. Pokud se to podaří zobrazí se místo nápisu MDC obraz z kamery.

Odpojení od serveru

Po připojení k serveru se nápis na tlačítku změní na Disconnect. Pro odpojení stačí kliknout na toto tlačítko.

Server

Konfigurace

Aplikace má nastavení uložená v souboru /usr/share/mdd/mdd.conf. Nastavení jsou následující:

- **width** - šířka obrazu v bodech (doporučeno 320)
- **height** - výška obrazu v bodech (doporučeno 240)
- **depth** - barevná hloubka (doporučeno 3)
- **device** - video zařízení z něhož se bude obraz snímat
- **motion_plug** - absolutní cesta k souboru modulu pro detekci pohybu
- **detection_mask_file** - absolutní cesta k souboru s maskou pro omezení oblastí detekce
- **capture_dir** - absolutní cesta k adresáři kam se mají ukládat obrázky a video pokud je detekován pohyb
- **capture_file** - absolutní cesta k souboru kam se ukládá obraz při každém načtení z kamery, může sloužit například jako web kamera do HTML stránky, pokud je prázdná nikam se neukládá. musí se jednat o obrázek typu JPEG!
- **add_date** - 0 - nedávat do obrazu datum, 1 - přidávat aktuální datum do obrazu
- **add_text** - 0 - nedávat do obrazu doplňující text, 1 - přidávat text ze souboru specifikovaného v text_file
- **text_file** - soubor s textem, který se umístí do obrázku, spolu s datem by neměl přesáhnout cca 30 znaků
- **text_color** - barva přidaného textu jako integer hodnota ve formátu 0x00rrggbb
- **save_video** - 0 - pokud je detekován pohyb, tak **ne**ukládat video, 1 - pokud je detekován pohyb, tak ukládat video
- **save_image** - 0 - pokud je detekován pohyb, tak **ne**ukládat obrázky, 1 - pokud je detekován pohyb, tak ukládat obrázky
- **server_port** - port na který se mohou hlásit klienti

- **authentization** - 0 - nepožadovat po klientech autentizaci, 1- vyžadovat autentizaci klientů před zasíláním dat

V souboru /usr/share/mdd/auth.conf jsou uložena data pro autentizaci ve formě

<jméno> <klíč>
<jméno> <klíč>
<jméno> <klíč>

Ukončení video serveru

Pokud je z nějakého důvodu potřeba ukončit video server sloužící pro zasílání dat klientům lze to provést zasláním signálu SIGUSR1 procesu mdd.

Znovunačtení konfiguračních souborů

Po změně hodnot v konfiguračním souboru je nutné aplikaci buď restartovat nebo jí zaslat signál SIGUSR2 aby si nové hodnoty ze souboru načetla.