VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SÍŤOVÝ INTERFACE K DETEKTORU KLÍČOVÝCH SLOV

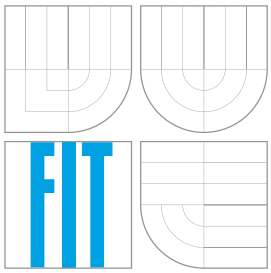
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN SKOTNICA

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SÍŤOVÝ INTERFACE K DETEKTORU KLÍČOVÝCH SLOV

NETWORK INTERFACE FOR KEYWORD SPOTTING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — MARTIN SKOTNICA
AUTHOR

VEDOUCÍ PRÁCE — Ing. IGOR SZÖKE
SUPERVISOR

BRNO 2007

Zadání

Síťový interface k detektoru klíčových slov

Network Interface for Keyword Spotting System

Vedoucí: Szöke Igor, Ing., UPGM FIT VUT

Oponent: Glembek Ondřej, Ing., UPGM FIT VUT

Zadání:
1. Seznamte se s toolkitem STK (toolkit pro rozpoznávání řeči) a s technikami síťové komunikace.
2. Navrhněte protokol a interface pro STK toolkit se zaměřením na část "detekce klíčových slov".
3. Implementujte aplikace klient-server která umožní jak on-line, tak i off-line komunikaci.
4. Otestujte funkčnost a zhodnoťte dosažené výsledky.

Část požadovaná pro obhajobu SP: Body 1, 2 a část bodu 3 ze zadání.

Kategorie: Softwarové inženýrství

Implementační jazyk: Perl, Matlab, Linux shell

Operační systém: Linux

Komerční software: Matlab

Literatura: Podle pokynů vedoucího.

Komentář: Cílem projektu je vytvořit TCP/IP interface k detektoru klíčových slov. Bude nějaka cílová aplikace která se připojí k serveru (detektor slov) a bude mu posílat data (záznam z mikrofonu nebo ze souboru) a server zpět bude vracet nalezená klíčová slova. Detektor je hotov.. jde jen o vytvoření interface a komunikace.

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologií Vysokého učeni technického v Brně.

Abstrakt

Rozpoznávání řeči je oblastí, která je v současné době celosvětově intenzivně studována. Systémy schopné rozpoznat mluvenou řeč se stávají součástí běžného života v mnoha aplikacích. Jednou z nich je i detekce klíčových slov, způsob jak odhalit výskyt určitých slov v datech. Detektor vyvinutý na VUT Fakultě informatiky nám umožňuje detekovat tyto slova. Cílem této práce je tedy navrhnout a implementovat síťový interface k detektoru klíčových slov na bázi klient/server. Cílová aplikace se připojí na server a posílá mu zvuková data. Server na tyto data spouští detektor klíčových slov a výsledek posílá zpět klientovi, kde se interaktivně zobrazí uživateli.

Klíčová slova

rozpoznávání řeči, detektor klíčových slov, STK toolkit, TCP, IP, soket, klient, server, zpracování zvuku, wxWidgets

Abstract

A considerable part of the research in computer science is dedicated to speech recognition as the speech-controlled systems become useful in many applications. One of them is the keyword spotting which makes possible to find words in audio data. Such a detector is developed at BUT Faculty of Information Technology. The goal of this work is to propose a network interface to this keyword detector based on client/server architecture. Client connects to the server and sends audio data. Server runs keyword detector with this received data and sends the result of keyword spotting back to client. Finally client visualizes the result and interact with user.

Keywords

speech processing, keywords spotting, STK toolkit, TCP, IP, socket, client, server, audio processing, wxWidgets

Citace

Martin Skotnica: Síťový interface k detektoru klíčových slov, diplomová práce, Brno, FIT VUT v Brně, 2007

Síťový interface k detektoru klíčových slov

Prohlášení

Prohlašuji, že jsem tento semestrální projekt včetně všech příloh vypracoval samostatně pod vedením Igora Szökeho a uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Skotnica
22. května 2007

Poděkování

Rád bych zde poděkoval svému vedoucímu za odborné vedení, poskytnuté rady a čas při řešení této diplomové práce.

Obsah

1 Úvod **3**

2 Rozpoznávání řeči **4**
2.1 Detekce klíčových slov . . . 4
2.2 Nástroje pro rozpoznávání řeči . . . 5
2.2.1 Vstupy a výstupy . . . 6

3 Síťová komunikace **7**
3.1 Architektura TCP/IP . . . 7
3.1.1 Spojované VS nespojované služby . . . 7
3.1.2 Packet . . . 8
3.1.3 IP . . . 9
3.1.4 TCP . . . 10
3.2 Sokety . . . 11
3.3 Zabezpečení komunikace . . . 12

4 Analýza požadavků **13**
4.1 Architektura . . . 13
4.2 Server . . . 14
4.2.1 Obsluha spojení . . . 14
4.2.2 Příjem dat od klienta . . . 14
4.2.3 Spouštění skriptů . . . 14
4.2.4 Odesílání dat klientovi . . . 14
4.3 Klient . . . 14
4.3.1 Komunikace se serverem . . . 15
4.3.2 Zpracování zvukových souborů . . . 15
4.3.3 Zpracování výstupu STK toolkitu . . . 15
4.3.4 Ovládání . . . 15
4.4 Technické požadavky . . . 16

5 Návrh **17**
5.1 Koncepce . . . 17
5.2 Komunikační protokol . . . 17
5.2.1 Příklad komunikace . . . 19
5.2.2 Přenos souboru . . . 19
5.3 Server . . . 20
5.3.1 Vlákna . . . 20
5.3.2 Volání STK toolkitu . . . 20

5.3.3 Bezpečnost . . . 22
5.4 Klient . . . 23
5.4.1 Zpracování zvuku . . . 23
5.4.2 Interpretace výstupu rozpoznávače . . . 24
5.4.3 Grafické uživatelské prostředí . . . 25
5.5 Konfigurovatelnost . . . 25

6 Implementace **27**
6.1 Výběr a popis technologií . . . 27
6.1.1 Použité knihovny . . . 28
6.2 Server . . . 29
6.3 Klient . . . 31
6.3.1 Třídy nízkoúrovňových funkcí . . . 31
6.3.2 Třídy a prvky grafického uživatelského prostředí . . . 32
6.3.3 Grafické uživatelské prostředí . . . 35

7 Rozšíření systému **36**
7.1 Online mód . . . 36
7.1.1 Rozšíření komunikačního protokolu . . . 36
7.1.2 Komunikace na protokolu UDP . . . 36

8 Závěr **37**

A Grafické uživatelské prostředí pod OS Linux **39**

B Skript pro spuštění STK toolkitu **40**

C Konfigurační soubory **41**

Kapitola 1

Úvod

Cílem této práce je navrhnout a implementovat síťový interface k detektoru klíčových slov, který by umožňoval rozpoznávání klíčových slov na nahraných zvukový datech.

Donedávna narážel vývoj umělé inteligence na nedokonalost při zpracování a porozumění počítačů přirozené lidské řeči. Vzniklo proto několik různých projektů a skupin řešící jak nejlépe zpracovat lidskou řeč. Jednou z nich je i skupina pro zpracování řeči Speech@FIT vedená na Fakultě informatiky, jenž vyvinula nástroj STK toolkit umožňující rozpoznávání řeči.

Výsledná aplikace by měla tedy sloužit k prezentaci práce skupiny Speech@FIT na STK toolkitu. Jelikož rozpoznávání řeči je náročné na výpočetní výkon a skladovou kapacitu řečových slovníků, navíc dodávat rozpoznávač a řečové slovníky přímo uživateli je z důvodu odcizení nebezpečné, není optimální provádět rozpoznávání u každého uživatele zvlášť. Je tudíž vhodné oddělit část výpočetní od části operativní. A právě to by měla zajistit cílová aplikace.

Nejlepší řešení se jeví vytvořit TCP/IP aplikaci na architektuře CLIENT/SERVER. Na výkonném počítači s rozpoznávačem slov se spustí server, který bude obsluhovat požadavky klientů na rozpoznávání. Server přijme data a přepošle je STK toolkitu. Jakmile je rozpoznávání dokončeno, server odešle klientovi rozpoznaná data a ten je uživateli graficky zobrazí.

Jelikož STK toolkit pro rozpoznávání řeči dovoluje více než jen detekci klíčových slov, můžeme provádět celou řadu dalších operací: převod řeči na text, identifikace mluvčího a rozpoznání mluvčího.

Téma této diplomové práce nenavazuje na žádné předchozí projekty, pouze na můj semestrální projekt.

Úvod k rozpoznávání řeči, popsán v druhé kapitole, udává možnosti zpracování a využití rozpoznávání řeči a práci STK toolkitu. Kapitola „analýza požadavků“ nám specifikuje a analyzuje základní požadavky na výslednou aplikaci.

V kapitole „návrh“ je uveden způsob síťové komunikace, návrh komunikačního protokolu a popis serverové a klientské části aplikace.

Implementace programu, výběr implementačních technologií, popis jednotlivých tříd a uživatelského rozhraní je podrobně uveden v kapitole „implementace“.

Kapitola 2

Rozpoznávání řeči

Rozpoznávání řeči je oblastí, která v současné době celosvětově podléhá intenzivnímu vývoji. Systémy schopné rozpoznat mluvenou řeč se stávají součástí běžného života v mnoha aplikacích. Jedná se především o automatizované informační systémy, ovládání různých zařízení hlasem, transkripce audio nahrávek, apod.

Obecně myslíme rozpoznáváním řeči postup, při kterém se podle určitých zaznamenaných vzorků zvuku rozpoznává, co bylo řečeno. Tato metoda rozpoznávání je založena na poznatcích o lidském slyšení. I když mají mluvčí stejný jazyk, liší se jejich hlasový projev právě v různé intonaci a zabarvení. Tyto rozdíly se snaží překonat složité výpočetní operace, takže proces rozpoznání řeči je velmi výpočetně náročný, zvláště jde-li o rozpoznávání plynulé řeči.

Rozpoznávání řeči lze rozdělit na několik oblastí:

- detekce klíčových slov - detekujeme výskyt zadaného slova
- převod řeči na text - převádíme mluvenou řeč na text
- rozpoznávání jazyka - rozpoznáváme zda jde o češtinu, angličtinu či jiný jazyk
- verifikace mluvčího - ověření zda opravdu mluví daná osoba
- identifikace mluvčího - identifikace osoby podle hlasu
- zpracování přirozeného jazyka - porozumění řeči v její komplexní podobě

Jelikož téma této diplomové práce spočívá v detekci klíčových slov, následující řádky popisují způsob jakým detektor klíčových slov pracuje.

2.1 Detekce klíčových slov

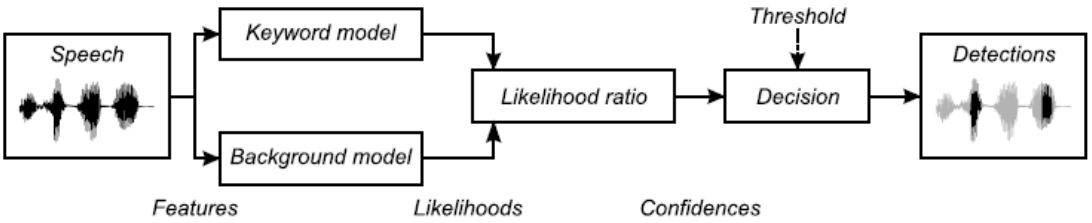
Je to způsob jak odhalit výskyt určitých slov v datech. Detekce klíčových slov v textu je velice jednoduchá, jde v podstatě pouze o prohledávání textového řetězce. Naproti tomu detekce slov ve zvukových datech není triviální, a to kvůli nepoměrně většímu objemu dat.

Informace je uložena ve změně akustiky a je vyjádřena konečným počtem způsobů. Nejprve je potřeba ze vstupního signálu extrahovat pouze potřebné informace. Odfiltrovat vliv prostředí, ve kterém mluví mluvčí, jeho barvu hlasu, vliv rychlosti řeči a další vlivy. Odfiltrovaný zvuk se klasifikátorem převede na logické jednotky hlasu, tzv. fonémy.

Existuje několik druhů klasifikátorů:

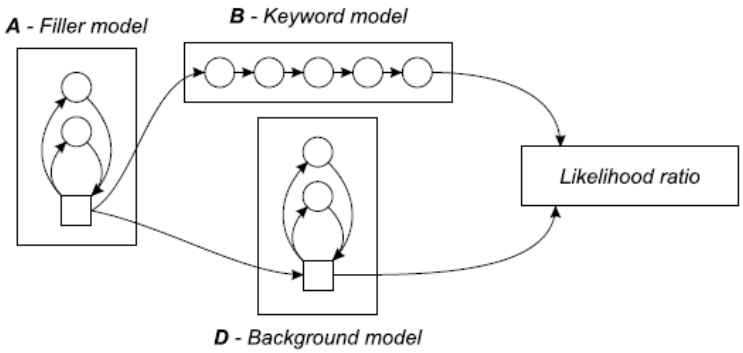
- Skryté Markovovy modely (HMM)
- Neuronové síťě
- a další

Signál řeči je parametrizován na vlastnosti řeči. Tyto vlastnosti jsou poté použity jako vstup modelů. Jedna část jako model klíčových slov a zbytek jako pozadí. Výstup modelů (klíčová slova a pozadí) je pravděpodobnost shody s modelem řeči. Poté pomocí prahování pravděpodobnosti získáme detekované klíčové slova [1].



Obrázek 2.1: Obecný model detektoru klíčových slov.

Akustická detekce klíčových slov používá modely fonémů, viz obrázek 2.2. Část A a C jsou tzv. plnící modely které modelují neklíčové části výrazu. Část B je lineární model daných klíčových slov. Část D je model pozadí který modeluje stejnou část výrazu jako model klíčových slov.



Obrázek 2.2: Obecný model detektoru klíčových slov.

2.2 Nástroje pro rozpoznávání řeči

Existuje již mnoho programů na rozpoznávání řeči. Hlavním je však HTK toolkit vyvíjený na univerzitě v Cambridge a STK toolkit vyvíjený na Fakultě informatiky VUT. Oba toolkity jsou nástroje pro statistické modelování řeči pomocí GM/HMM. STK obsahuje navíc extrakci příznaků a parametrů řeči, které jsou pak klasifikovány klasifikátory jako HMM a nebo NN a je proto mnohem komplikovanější.

Jak HTK tak i STK toolkit obsahuje trénovaní a rozpoznávání pomocí GM/HMM. Oproti HTK obsahuje STK toolkit navíc pokročilé techniky trénovaní HMM (diskriminativní trénovaní) a také trénovaní a rozpoznávání pomocí neuronových sítí.

Oba toolkity jsou zaměřeny spíše pro experimentální vývoj. Obsahují několik programů, které mezi sebou komunikují pomocí textových souborů. Celý systém je tedy slepen jako „lego“. Bohužel zatím nepodporují žádnou síťovou komunikaci, na kterou by se dal přímo navázat klient. Proto musíme vytvořit síťovou mezivrstvu mezi klientem a STK toolkitem, která by nám umožňovala přenos požadavků na rozpoznávání.

Tím by měl být server, který bude přijímat požadavky klienta a na jeho žádost bude volat script, který spustí příslušnou sekvenci příkazů STK toolkitu a vrátí výsledky v podobě textových souborů. Tyto výsledky pak server pošle zpět klientovi, který je uživateli zobrazí.

2.2.1 Vstupy a výstupy

Vstupem STK toolkitu je řeč, řečový slovník a natrénovaná neuronová síť. Výstup je pak soubor obsahující rozpoznaná slova, jejich pozici ve zvukovém souboru a pravděpodobnost výskytu jednotlivých slov při rozpoznávání.

STK toolkit podporuje základní zvukové formáty `raw` a `wav`. Výstup je pak ukládán do souboru s příponou `.mlf`, definující počáteční a koncový čas nalezených slov, jednotlivá slova a jejich pravděpodobnost výskytu. Formát výstupního souboru je popsán na následujícím příkladu:

```
14500000 17200000 ahoj -9.6195
17200000 21300000 hanka -24.4331
```

Jelikož je STK toolkit sada programů, musíme při rozpoznávání zajistit jejich volání ve správném pořadí. To nám umožňují skripty jimž budeme předávat pouze základní parametry.

Detekce klíčových slov

Pro detekci klíčových slov potřebujeme několik parametrů. Vstupem je tedy zvukový soubor, soubor klíčových slov a práh, určující minimální pravděpodobnost výskytu. Jestliže nebudeme práh zadávat, vrátí nám detektor všechny detekovaná slova. To nám umožní filtrovat nalezená slova až na straně klienta.

Převod řeči na text

Vstupem STK toolkitu při převodu řeči na text je zvukový soubor. Rozpoznání pracuje ve dvou módech. Mód úplného rozpoznání, kdy detektor vrátí co si myslí že tam je, nebo mód zarovnání, kdy zadá uživatel co tam je a systém mu řekne s jakou pravděpodobností to tam je.

Pokud tedy nechceme řeč převést na text, ale pouze zarovnat na řeč, tak vstupem musí být i transkripce.

Rozpoznání jazyka řečníka a identifikace mluvčího

Rozpoznání jazyka řečníka a identifikace mluvčího pracují také v obou módech. Vstupem je zvukový soubor a pokud chceme zarovnávat, tak i jazyk či jméno mluvčího.

Kapitola 3

Síťová komunikace

V části síťová komunikace jsou popsány základní komunikační protokoly použité v této práci. Podkapitola „architektura TCP/IP“ popisuje základní komunikační principy na protokolu TCP/IP. V části „spojované VS nespojované služby“ se dozvíme, které služba je vhodné používat na daný typ aplikace. Části „paket“, „TCP“, „IP“ a „socket“ nás uvedou do problematiky komunikace pomocí TCP/IP.

3.1 Architektura TCP/IP

Protokolová architektura TCP/IP je definována sadou protokolů pro komunikaci v síti. Komunikační protokol je množina pravidel, které určují syntaxi a význam jednotlivých zpráv při komunikaci.

Vzhledem ke složitosti problémů je síťová komunikace rozdělena do tzv. vrstev, které znázorňují hierarchii činností. Každá vrstva využívá služeb vrstvy nižší a poskytuje své služby vrstvě vyšší. Komunikace mezi stejnými vrstvami dvou různých systémů je řízena komunikačním protokolem za použití spojení vytvořeného sousední nižší vrstvou. Architektura TCP/IP je členěna do čtyř vrstev:

Vrstva síťového rozhraní umožňuje přístup k fyzickému přenosovému médiu (skrze MAC adresu).

Síťová vrstva zajišťuje síťovou adresaci, směrování a předávání datagramů. Je na ní postaven protokol **IP**.

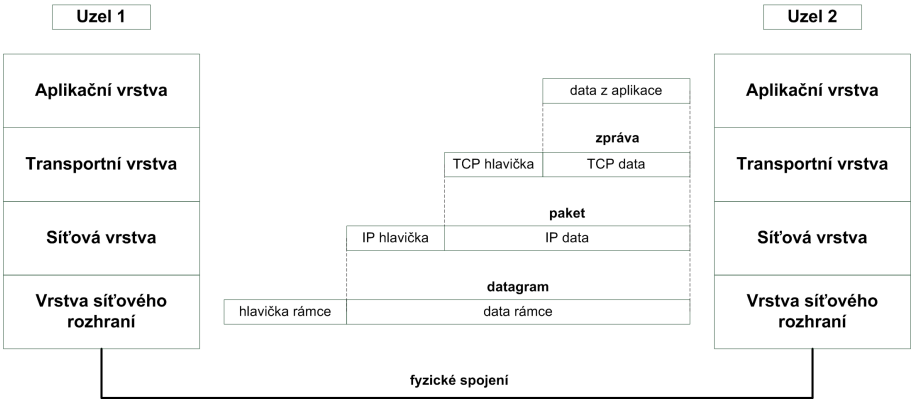
Transportní vrstva poskytuje spojované (protokol **TCP**) a nespojované (protokol **UDP**) transportní služby.

Aplikační vrstvu využívají programy pro přenos dat po síti. Obsahuje rodinu přenosových protokolu jako HTTP, FTP, Telnet, POP3 a další.

Pro rozlišení aplikačních protokolů se používají tzv. porty, což jsou domluvená číselná označení pro síťové aplikace. Každé síťové spojení je jednoznačně určeno transportním protokolem a číslem portu.

3.1.1 Spojované VS nespojované služby

Podle činnosti kterou chceme na síti provádět, používáme spojované nebo nespojované služby. V některých případech je vhodnější použit spojovaných služeb, v jiných zase ne-



Obrázek 3.1: Komunikace v modelu TCP/IP.

spojovaných. Protože není rozumné posílat velké bloky dat přes síť v jedné zprávě, musí se data rozdělit na menší kousky. Jejich hlavní rozdíl je tedy, jak už plyne z jejich názvu, postup při odesílání a přijímání těchto jednotlivých částí.

Spojované služby zajišťují přijímání jednotlivých částí dat ve správném pořadí a při ztrátě obstarávají jejich nové odeslání.

Naproti tomu nespojované služby správné a bezchybné doručení nezajímá. Můžeme jich tedy použít pouze tam, kde nepotřebujeme spolehlivé doručení všech dat. Jejich výhodou je hlavně větší kontrola komunikace a také to, že nezahlcují zbytečně síť při větším objemu dat, protože nemusí potvrzovat přijetí jednotlivých paketů.

- **Spojované (connection-oriented) služby**
  - zaručeno doručení datagramů ve správném pořadí (stream)
  - aplikace je jednodušší, ale nemůže řídit komunikaci
  - obdoba telefonního spojení
  - implementace v protokolu TCP je komplikovaná
- **Nespojované (connectionless) služby**
  - není zaručeno pořadí ani doručení datagramů
  - kontrolu musí provádět aplikace
  - aplikace může lépe řídit komunikaci
  - obdoba poštovního spojení
  - implementace v protokolu UDP je jednodušší

Jelikož daná aplikace vyžaduje pro přenos dat spolehlivé doručování, je vhodné použít spojované služby.

3.1.2 Packet

Packet je jednotka/blok, po které se odesílají data. V sítích se nic neposílá po 1 bajtu, ale právě po paketech. Pokud se cela data (blok bajtů), která chceme odeslat jako celek, nevejdou do 1 paketu, dojde k tzv. fragmentaci, což znamená, že se daný blok bajtů rozdělí do několika paketů, které se postupně odešlou.

Během trvání spojení je obvykle odesláno poměrně velké množství paketů. V některých druzích sítí, nemusí ani všechny pakety proběhnout přes stejnou cestu, ale mohou být přesměrovány na rychlejší či výhodnější spojení.

Paket se skládá ze tří základních prvků:

- **hlavička** - informace potřebné pro doručení paketu do místa určení
- **datová oblast** - vlastní vyslané informace
- **trailer** - potvrzení bezchybnosti přenosu

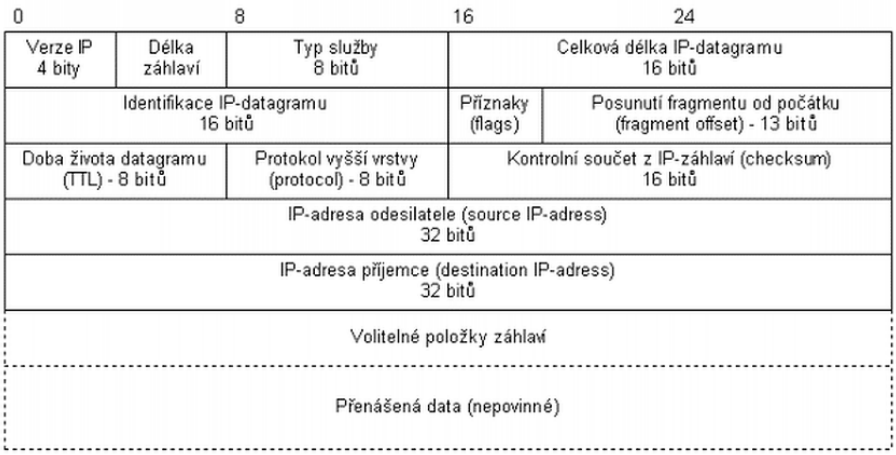
Dobrým přirovnáním může být dopis – hlavička je jako adresa odesílatele a příjemce. Data jsou to, co se nachází uvnitř obálky.

3.1.3 IP

Internet Protocol je základní protokol síťové vrstvy a celého Internetu. Provádí vysílání datagramů na základě síťových IP adres obsažených v jejich záhlaví. Poskytuje vyšším vrstvám síťovou službu bez spojení, tzn. že vyšším vrstvám předává data tak, jak byla přijata ze sítě a tedy i v nesprávném pořadí. Vyšší vrstvy si jednotlivé datagramy již musí spojit samy.

Tento protokol se dále stará o segmentaci a znovusestavení datagramů do a z rámců podle protokolu nižší vrstvy.

Každý datagram je samostatná datová jednotka, která obsahuje všechny potřebné údaje o adresátovi, odesílateli a pořadovém čísle datagramu ve zprávě. Datagramy putují sítí nezávisle na sobě a pořadí jejich doručení nemusí odpovídat pořadí ve zprávě. Jelikož není doručení datagramu druhé straně zaručeno, musí si tuto spolehlivost zajistit některá z vyšších vrstev (např. protokol TCP).



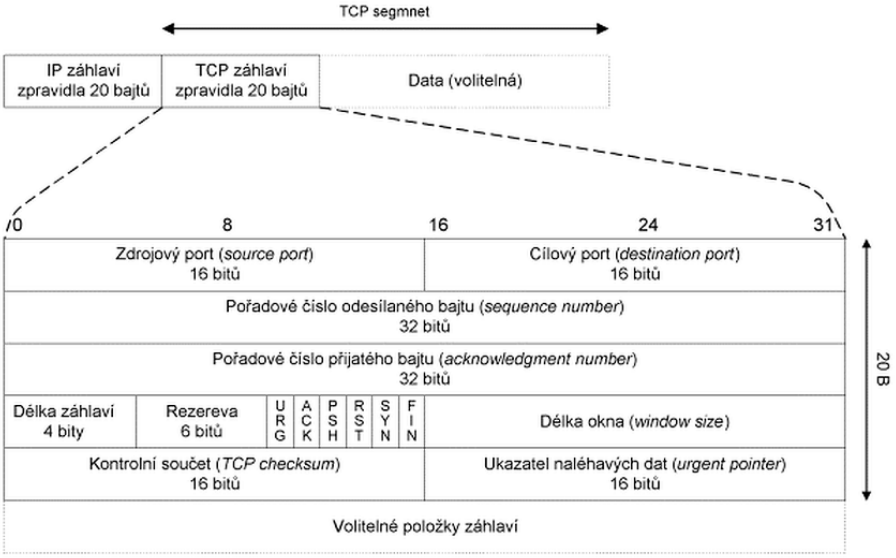
Obrázek 3.2: Schéma datagramu IP.

V současné době existují dvě verze protokolu, IPv4 a IPv6. Zatím je převážně používána starší verze IPv4. Nová verze IPv6 řeší nedostatek adres v IPv4, bezpečnostní problémy a vylepšuje další vlastnosti protokolu IP.

3.1.4 TCP

Protokol TCP [2] je spojově orientovaný protokol pro přenos toku dat na transportní vrstvě se spolehlivým doručováním. TCP je prostřední vrstvou mezi IP protokolem pod ním a aplikací nad ním.

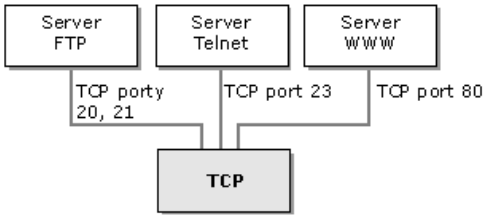
Síťové aplikace ke vzájemné komunikaci využívají spolehlivé spojení na způsob tzv. roury (streamy), kterou proudí proud dat (podobně jako voda ve vodovodní trubce). IP protokol takové streamy neposkytuje, ten posílá pouze nespolehlivé pakety. Tedy TCP protokol používá služby IP protokolu opakovaným odesíláním nespolehlivých paketů při ztrátě paketu, tím zajišťuje spolehlivost. TCP ověří, že se pakety neztratily tím, že každému paketu přidělil číslo sekvence, které se také použije k ověření, že data byla přijata ve správném pořadí. Přeuspořádáváním přijatých paketů pak zajišťuje správné pořadí paketů.



Obrázek 3.3: Hlavicka TCP.

TCP protokol na straně příjemce posílá zpět potvrzení pro pakety které byly úspěšně přijaty. Pokud by se odesílateli potvrzení nevrátilo do rozumné doby, označil by data za ztracená a vyslal by je znovu.

K rozlišení komunikujících aplikací používá TCP protokol čísla portů. Každá strana TCP spojení má přidružený 16bitové bezznaménkové číslo portu (maximálně 65535 portů) přidělené aplikaci. Porty jsou rozčleněny do třech skupin: dobře známé, registrované a dynamické/privátní.



Obrázek 3.4: TCP porty.

Pro mnoho aplikací není TCP vhodné. Velkým problémem je (alespoň u normálních implementací), že aplikace po ztrátě jednoho paketu nemůže dostat následující pakety do té doby, dokud není ztracený paket znovu poslán a úspěšně přijat.

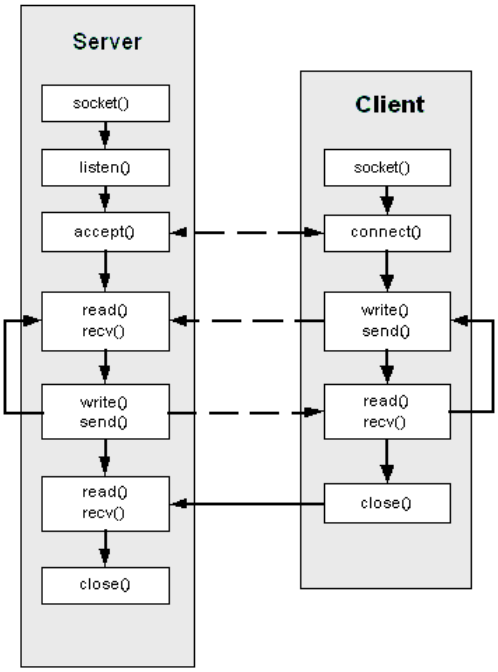
Tam, kde je TCP nevhodné, se často používá UDP, které poskytuje aplikaci kontrolu nad multiplexováním a ověřováním kontrolních součtů. Zato ale UDP neprovádí fragmentaci proudu dat do paketů a zpátky jejich rekonstruování, ani opětovné posílání ztracených paketů. Tyto operace jsou plně ponechány na koncové aplikaci.

3.2 Sokety

Soket je obecný mechanismus pro komunikaci, který byl navržen na počátku 80. let v Berkeley během implementace TCP/IP do BSD Unix a postupně se rozšířil nejen do ostatních odnoží Unixu, ale také do operačních systémů MacOS a MS Windows. Tento mechanismus lze v zásadě používat pro komunikaci procesů, vláken, či různých protokolů, v našem případě tedy pro komunikaci protokolu TCP/IP.

Představíme-li si komunikaci jako rouru, kterou tečou data, socket by bylo pojmenování pro její konce. Při komunikaci si každý proces vytvoří svůj socket a nastaví jeho parametry tak, aby pomocí něj mohl komunikovat se soketem jiného procesu (na jiném počítači).

Pomocí Soketového API [3] je možno využívat různé síťové protokoly. Toto API je stejné pro všechny operační systémy Unixového typu, tedy na všech těchto systémech by měly být k dispozici stejné funkce se stejnými parametry. Také na platformě Windows existuje soketové API (WinSock2), avšak s menšími odlišnostmi v implementaci jednotlivých funkcí.



Obrázek 3.5: Diagram komunikace pomocí soketů.

3.3 Zabezpečení komunikace

Jelikož se v počítačových sítích můžou vyskytnout chyby, je potřeba každý program pracující se síti proti nim zabezpečit. Opravu chyb při komunikaci a při ztrátě paketů zajišťuje protokol TCP.

Aplikaci je třeba zabezpečit i před zneužitím a před útokem hackerů. Lze využívat chyb programů k získání přístupových práv administrátora a tím nabourání se do celého systému. Poměrně častou technikou je přetečení bufferu (buffer-overflow). Útočník podstrčí neošetřené aplikaci více dat než je schopna si zapsat do zásobníku a tím dojde k přepsání části kódu v paměti za zásobníkem. Tímto způsobem může útočník systému podstrčit vlastní kód, který je pak za jistých okolností spuštěn a útočník tak může lehce získat práva administrátora.

Již trochu méně nebezpečné jsou útoky typu DoS (Denial of Service), neboli „likvidace služby“. Cílem těchto útoků je zahlcení cílového počítače tak, aby přestal poskytovat danou službu (Web Server, DNS server, atd.). Pro tento typ útoků je charakteristické, že je prováděn z většího počtu počítačů najednou. Útočník se systematicky snaží posílat více požadavků k obsluze cílové aplikaci než je ona schopna obsloužit. Tím dojde k omezení, nebo výpadku služby.

Typy síťových útoků:

- **DoS – Denial of Service (likvidace služby)**
  - Zahlcení, nebo poškození cílového počítače.
  - SYN flood, vyčerpání počtu připojení, buffer overflow.
- **Získání přístupu na cílový počítač**
  - Většinou snaha spustit vlastní kód na cílovém počítači (buď jako samostatný proces, nebo umístit kód do procesu serveru).
  - Snaha o spuštění kódu pod právy administrátora.
  - Buffer overflow, remote code execution.

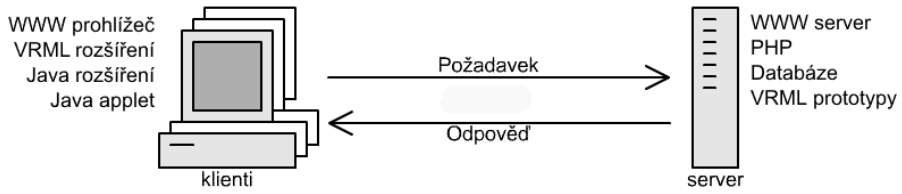
Kapitola 4

Analýza požadavků

V této části jsou představeny funkce a vlastnosti, které by měl navrhovaný program obsahovat, aby plnil své cíle.

4.1 Architektura

Ze zadání plyne, že aplikace má být postavena na síťové architektuře CLIENT/SERVER. Tato architektura rozděluje aplikaci na dva bloky. Na server, který poskytuje služby a na klienta, který je konzumuje.



Obrázek 4.1: Architektura server/klient.

Výsledné aplikace musí být tedy dvě, každá s jiným požadovanými vlastnostmi.

- **Charakteristika serveru:**
  - Pasivní
  - Čeká na požadavky od klienta
  - Při přijetí požadavku jej obslouží
- **Charakteristika klienta:**
  - Aktivní
  - Posílá požadavky serveru
  - Čeká na odpovědi

4.2 Server

Na server klademe několik požadavků. Kromě přenosu dat mezi serverem a klientem, je to i způsob obsluhy spojení s jednotlivými klienty. Hlavním cílem je ovšem rozpoznávání zvukových dat. Aby jsme tuto funkčnost mohli zajistit, musíme nějakým způsobem provázat server s rozpoznávačem.

Server musí mít tedy několik základních vlastností, jež jsou popsané v následujících podkapitolách.

4.2.1 Obsluha spojení

Obsluhu spojení lze naimplementovat dvěma způsoby. Jedním je, že server obsluhuje pouze jednoho spojení s klientem a ostatní klienti musí čekat dokud server takto připojeného klienta neobslouží. Taková obsluha spojení je pro většinu síťových aplikací nevhodná, protože se tím ztrácí hlavní výhoda architektury Klient/Server a to distribuce služby kterou poskytuje server mezi více klientů.

Druhý způsob je tedy výhodnější, protože umožňuje, aby bylo ve stejný čas připojeno více klientů najednou.

4.2.2 Příjem dat od klienta

Server musí také zvládat příjem dat od klienta. Jedná se především o příjem zvukových a textových dat, které se budou ukládat v podobě souborů, aby je mohly dále zpracovat skripty spouštějící STK toolkit (viz. kapitola 2.2.1).

4.2.3 Spouštění skriptů

Hlavním úkolem serveru a celé aplikace je využití funkcí STK toolkitu. Jak jsme si řekni v kapitole 2.2, rozpoznávač je pouze sada spustitelných souborů s mnoha parametry nastavení pro spuštění (nastavení neuronové sítě, parametry zvukového souboru a jiné). Proto musí být na straně serveru umístěny spouštěcí skripty.

Server bude tedy spouštět pouze tyto skripty a předávat jim název zvukového souboru, souboru klíčových slov a souboru pro výstup.

4.2.4 Odesílání dat klientovi

Zpět klientovi bude server odesílat výstup z rozpoznávače. Při výskytu jakékoliv chyby, která nastane při rozpoznávání, se musí odeslat informace klientovi o tom, že chyba nastala a také její obsah.

Aby mohl klient zobrazit uživateli které módy rozpoznávání jsou dostupné (Keyword spotting, Speech to text, atd.), musí server informovat klienta o nastavení rozpoznávače.

Další vlastnost serveru musí být odesílání klientovi standardní nastavení klienta, např. nastavení barev zobrazení, které je uloženo na serveru.

4.3 Klient

Hlavní aplikací této diplomové práce je klient. Zajišťuje spojení se serverem, předává mu zvukové soubory, zobrazuje výsledky STK toolkitu a zajišťuje interaktivitu s uživatelem pomocí grafického uživatelského prostředí (GUI).

4.3.1 Komunikace se serverem

Požadavky na komunikaci serveru s klientem byla popsána v kapitole 4.2, obdobně bude probíhat komunikace klienta se serverem. Klient se musí umět připojit k serveru, vyžádat si od něj nastavení STK toolkitu a podle něj zobrazit uživateli možnosti rozpoznávání.

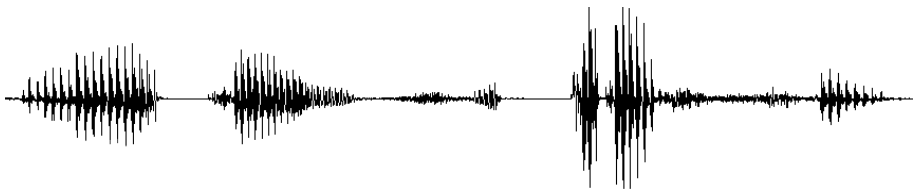
Dále musí umět odesílat na server zvukový soubor a soubor klíčových slov.

4.3.2 Zpracování zvukových souborů

Pod pojmem „zpracování zvukových souborů“ máme na mysli jejich čtení, přehledné zobrazení a přehrávání.

Jelikož existuje mnoho různých formátů zvukových souborů, které se liší jak ve způsobu uložení dat (integer/float) tak i ve způsobu komprese (ztrátová - `mp3` / bezztrátová - `wav`), není hlavním požadavkem schopnost číst co největší počet těchto formátů. STK toolkit pracuje pouze s formátem `raw` a `wav`, tedy hlavní důraz při čtení zvukových souborů je kladen na tyto formáty.

Jelikož jsou ve zvukovém souboru data uložena jako posloupnost vzorků, můžeme tyto vzorky pospojovat a tím zobrazit jeho signál. Klient musí zajistit funkce tzv. přibližování, vzdalování („zoom in“ a „zoom out“) a posuv zobrazeného signálu.



Obrázek 4.2: Zobrazený signál zvuku.

Klient musí umět přehrávat celý zvukový soubor i pouze jednotlivé rozpoznané části.

4.3.3 Zpracování výstupu STK toolkitu

Jakmile klient přijme ze serveru výstup z rozpoznávače, musí jej uživateli přehledně zobrazit. Jestliže nedojde k rozpoznání nebo nastane jiná chyba při rozpoznávání, server pošle informaci o chybě a klient o ní informuje uživatele.

Výstup z STK toolkitu je seznam nalezených slov, jejich pozic a pravděpodobnosti výskytu ve zvukovém souboru. Klient musí zobrazit všechny tyto tři údaje.

4.3.4 Ovládání

Obecné požadavky na ovládání programu klienta jsou:

- **Jednoduchost** spočívá ve zobrazení pouze nezbytných funkcí a informací, které uživatel ke své práci potřebuje.
- **Přehlednost** znamená vhodně zvolený způsob rozložení ovládacích prvků a vhodné použití barev při vykreslování.
- **Interaktivita** umožňuje uživateli zasahovat do zobrazování informací a za běhu programu je měnit.

4.4 Technické požadavky

Na obě aplikace klademe také několik technických požadavků:

- **Multiplatformnost** spočívá v možnosti spustit aplikaci na více operačních systémech, bez úprav zdrojových kódu.
- **Maximální konfigurovatelnost** znamená, že klient i server si budou ukládat co nejvíce svých možných nastavení do souboru, kde je bude moci uživatel upravovat.
- **Otevřené zdrojové kódy** umožňují komukoliv upravovat jejich obsah, bez nutnosti pracného získání oprávnění či souhlasu autora. Jedná se o licence typu Open Software.
- **Ošetření proti útokům** by měla mít každá síťová aplikace (hlavně server), aby se zamezilo zneužití aplikace k proniknutí do systému na kterém je spuštěna, nebo aby nebylo znemožněno poskytování služby.

Kapitola 5

Návrh

V této části je popsán návrh serveru a klienta a síťové komunikace mezi nimi. Ta je založena na protokolu TCP/IP a pro samotnou komunikaci využívá technologii soketů. V klientské části je popsán návrh grafického uživatelského prostředí.

5.1 Koncepce

Jak už bylo zmíněno, cílem této práce je navrhnout a implementovat síťový interface k detektoru klíčových slov.

Pod pojmem „síťový interface“ je třeba si představit jak ovládání programu, tzv. grafické uživatelské prostředí (GUI), tak i to co se skrývá pod pod povrchem, tedy metody komunikace, způsob jakým se obě aplikace budou domlouvat na jednotlivých operacích a v neposlední řadě také souborové operace se zvukovými soubory.

Jelikož STK toolkit zatím nepodporuje online rozpoznávání a v době návrhu i implementace nebylo známo rozhraní, návrh i implementace se zakládá pouze na offline rozpoznávání.

5.2 Komunikační protokol

Aby mohli server s klientem spolu komunikovat, musí být mezi nimi stanovena jistá pravidla, kterými se budou obě aplikace řídit při vzájemné komunikaci.

Definujeme tzv. komunikační protokol. Ten specifikuje způsob odesílání a přijímání dat, tak aby si obě strany, tedy server a klient, navzájem rozuměli.

Číslo příkazu	**Příkaz**	**Význam**
0	WAV	přenos zvukového souboru
1	KWS	přenos souboru klíčových slov
2	PROCESS	spuštění STK toolkitu
3	MLF	přenos výstupu z STK toolkitu
4	SERVER CONFIG	přenos vlastností serveru
5	CLIENT CONFIG	přenos vlastností klienta
6	ERROR	přenos zprávy o chybě

Tabulka 5.1: Seznam akcí komunikačního protokolu.

Komunikační protokol je založen na technologii soketů, popsaných v kapitole 3.2. Skládá z několika příkazů definující typ akce, která se má provést. Každý příkaz je vlastně číslo

(konstanta), které musí mít v obou aplikacích pro jeden příkaz stejnou hodnotu. Podle hodnoty tohoto čísla pozná klient a server, jaká akce má být provedena. Velikost přenášeného čísla musí být v obou aplikacích také stejné.

Tabulka 5.1 definuje hlavní příkazy komunikačního protokolu.

Příkaz WAV

K přenosu zvukového souboru slouží příkaz **WAV**. Ke zpracování a rozpoznání zvukového souboru dochází pouze na serveru, tudíž ho je potřeba přenášet pouze z klienta na server.

Příkaz KWS

Slouží k přenosu klíčových slov. Chová se obdobně jako příkaz **WAV**, tj. přenos probíhá pouze z klienta na server.

Příkaz PROCESS

Tento příkaz posílá klient serveru. Informuje jím server o tom, že má spustit proces rozpoznávání na STK rozpoznávači. Aby mohl server určit, kterou akci má na rozpoznávači spustit, musí poslat klient serveru další informaci.

Jedná se o jeden z množiny příkazů akcí rozpoznávače, popsaných v tabulce 5.2. Podle tohoto příkazu spustí server na STK toolkitu danou akci. Za číslem akce se odešle ještě číslo konfigurace, se kterou má rozpoznávač pracovat.

Číslo příkazu	Příkaz	Význam
7	STK KWS	spuštění procesu rozpoznávání klíčových slov
8	STK STT	spuštění procesu převodu řeči na text
9	STK LID	spuštění procesu identifikace jazyka
10	STK SPK	spuštění procesu identifikace mluvčího

Tabulka 5.2: Seznam akcí pro STK toolkit.

Příkaz MLF

Po úspěšném dokončení rozpoznávacího procesu, je potřeba přenést výsledky rozpoznávání zpět klientovi, aby je ten mohl uživateli zobrazit. K tomu slouží příkaz **MLF**. Logicky, přenos výsledků probíhá pouze ze serveru na klienta.

Příkaz SERVER CONFIG

Protože rozpoznávač nemusí nutně zvládat všechny typy procesu rozpoznávání, musí se nějakým způsobem zamezit žádosti klienta o process rozpoznávání, který server neumí. Klient tedy potřebuje znát schopnosti STK toolkitu. K tomu slouží příkaz **SERVER CONFIG**.

Pomocí tohoto příkazu se pošle klientovi informace o schopnostech rozpoznávače a jeho konfiguracích. Ten jí poté zobrazí uživateli a zamezí spuštění těch rozpoznávacích procesů, které nejsou podporovány.

Příkaz CLIENT CONFIG

Jedním z požadavků na klienta je i možnost stažení přednastavené konfigurace klienta ze serveru. To zajišťuje příkaz **CLIENT CONFIG**.

Příkaz ERROR

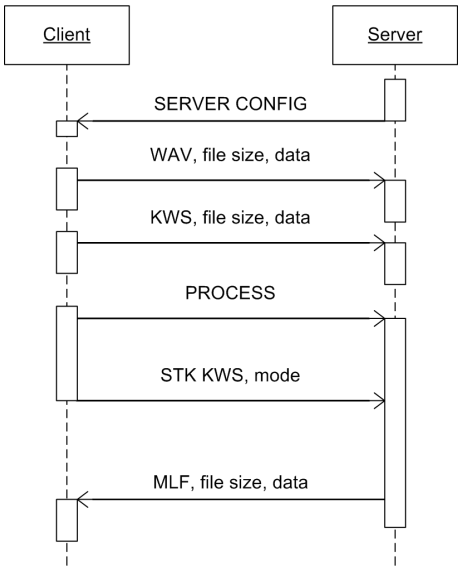
Při rozpoznávacím procesu může na straně serveru dojít k chybě. Příkaz **ERROR** umožňuje poslat klientovi informaci o tom, že chyba nastala a odeslat mu její obsah (zprávu o chybě).

5.2.1 Příklad komunikace

Na názorném příkladě si ukážeme vzájemnou komunikaci mezi serverem a klientem při rozpoznávání klíčových slov.

Po připojení klienta k serveru, mu server pomocí příkazu **SERVER CONFIG** pošle zpět nastavení schopností STK toolkitu. Uživatel provede v klientovi potřebné nastavení, tj. načte zvukový soubor, nastaví akci pro rozpoznávač a zadá klíčová slova. Poté pošle serveru příkaz **WAV** a **KWS**, čímž zajistí přenesení potřebných souborů. Server přijaté soubory uloží na disk. Po odeslání příkazu **PROCESS** bude následovat příkaz **STK KWS**, čímž řekneme serveru, že má spustit rozpoznávání klíčových slov. Výsledek pak pošle klientovi pomocí příkazu **MLF**.

Tento příklad je znázorněn na obrázku 5.1.



Obrázek 5.1: Příklad komunikace.

5.2.2 Přenos souboru

Nejdůležitější funkcí v komunikaci je přenos souboru mezi klientem a serverem. Přenos probíhá následovně:

1. Pošle se číslo, značící velikost přenášeného souboru.
2. Následuje přenos vlastních dat

3. Jakmile přijímací strana přijme ze sítě tolik dat, kolik byla velikost přenášeného souboru, přenos souboru se ukončí.

Hlavička posílaná při přenosu souboru je popsána v tabulce 5.3.

Velikost	**Obsah**
4 bajty	velikost souboru
[velikost souboru]	data

Tabulka 5.3: Hlavička odesílaná při přenosu souboru.

5.3 Server

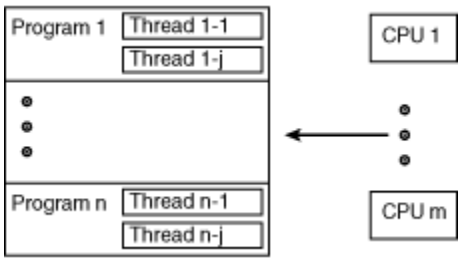
V kapitole 4.2 jsme si určili požadavky na funkce serveru. Server tedy bude obsluhovat spojení, přijímat od klienta zvuková data a soubor klíčových slov, spouštět rozpoznávač a z něj odesílat výstup zpět klientovi.

V kapitole 5.2 jsme specifikovali komunikační protokol, který definuje způsob komunikace mezi serverem a klientem. Zbývá nám tedy popsat, jakým způsobem bude server obsluhovat spojení a spouštět rozpoznávač.

5.3.1 Vlákna

Jedním z požadavků na obsluhu spojení je možnost obsluhovat více klientů najednou. To zajistíme pomocí technologie vláken.

Vlákno (Thread) označuje posloupnost po sobě jdoucích operací. Každá spuštěná aplikace má alespoň jeden proces a každý proces má alespoň jedno vlákno, ve kterém běží. Program s více vlákny označujeme jako multithreadový, tedy že uvnitř jednoho procesu v jednom adresovém prostoru se sdílenou pamětí, může zároveň běžet více vláken.



Obrázek 5.2: Multithreading.

Každý pokus o navázání spojení klienta se serverem vytvoří samostatné vlákno, které pokračuje v komunikaci pouze s daným klientem, dokud se klient neodpojí. Jelikož vlákno udržující spojení běží nezávisle na hlavním vlákně procesu, může server v hlavním vlákně čekat na spojení s dalším klientem. Tím je zajištěno spojení s více klienty najednou.

5.3.2 Volání STK toolkitu

Jestliže vlákno serveru přijme příkaz **PROCESS** následovaný jedním z typu akcí, spustí v samostatném procesu STK toolkit. Vlákno poté čeká, dokud nově vytvořený proces nevrátí

vláknu zpět řízení, tzn. dokud nedokončí STK toolkit svou činnost. Výsledek rozpoznávání se poté musí zpracovat a poslat zpět klientovi.

Jak bylo popsáno v kapitole 2.2 STK toolkit není pouze jeden program, ale je to sada aplikací, kde každá má svou specifickou úlohu v rozpoznávacím procesu (práce s řečovým slovníkem, s natrénovanou neuronovou sítí). Proto je vhodné vytvořit několik skriptů (pro každý typ rozpoznávání jeden), které budou postupně spouštět jednotlivé aplikace.

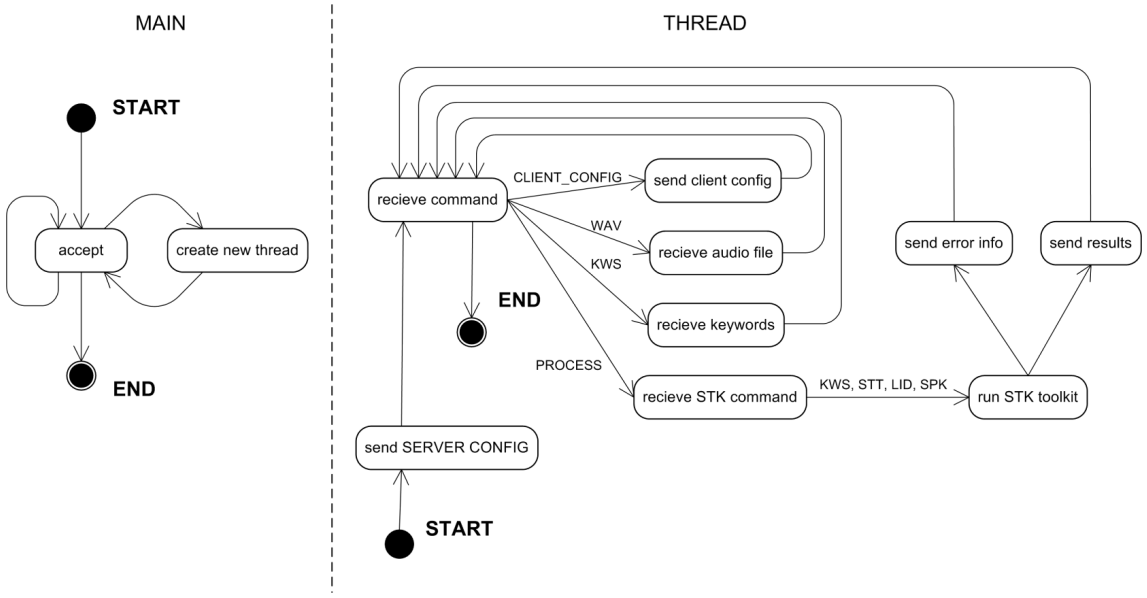
Skripty musí mít několik vstupních parametrů. Jejich počet se liší pouze v požadavku na soubor klíčových slov. Ostatní parametry zůstávají stejné.

Typ akce	**Parametry**
Rozpoznávání klíčových slov	zvukový soubor a soubor klíčových slov
Převod řeči na text	zvukový soubor
Identifikace jazyka	zvukový soubor
Identifikace mluvčího	zvukový soubor

Tabulka 5.4: Parametry jednotlivých skriptů STK rozpoznávače.

Jelikož při rozpoznávání by mohla nastat chyba, je vhodné aby server o tom mohl informovat uživatele na straně klienta. Skripty tedy musí vracet také informaci o chybě pokud nějaká nastala. Protože skripty normálně vypisují chyby na chybový výstup, můžeme tento výstup přesměrovat do souboru a číst ho odtud. Server bude po dokončení rozpoznávání hledat tento soubor s chybovým výstupem a pokud existuje, pošle se jeho obsah zpět klientovi a ten o ní informuje uživatele.

Obrázek 5.3 znázorňuje model fungování serveru.



Obrázek 5.3: Model serveru.

5.3.3 Bezpečnost

Zajistit bezpečnost komunikace a fungování serveru je určitě jednou z důležitých vlastností programu. V kapitole 3.3 jsme popsali nejčastější typy síťových útoků, kterými lze programy napadnout a získat tak přístup administrátora, či zlikvidovat službu poskytovanou serverem. Z kapitoly vyplývá, že nejčastějšími útoky jsou útoky „přetečení buferu“ (buffer overflow) a „likvidace služby“ (Denial of Service).

Buffer overflow

Základní ochrana proti přetečení buferu se musí zajistit na úrovni komunikačních příkazů. Při žádném z volání příkazů sloužícím k příjmu dat ze sítě, nesmí dojít k přetečení alokovaného prostoru proměnné do které se data ukládají, tzn. že v žádném případě nesmí dojít k načtení více dat ze sítě než je skutečně potřeba. To proto, aby nedošlo k přepsání adresového prostoru mimo adresový prostor alokované proměnné. Zabezpečit tento požadavek lze pomocí přísné kontroly alokace proměnných podle velikostí přesouvaných dat.

Denial of Service

Při ochraně proti útoku typu likvidace služby, musíme postupovat odlišně. Pro tento útok je charakteristické, že se útočník snaží z různých adres zahltit server mnoha nesmyslnými požadavky, jenž se je snaží obsluhovat a nezbývá mu čas na obsluhu „normálních“ klientů. Komplexní zabezpečení proti tomuto typu útoku není triviální a jeho řešení je spíše na celou diplomovou práci. Proto si vystačíme pouze se základním řešením a to, že dovolíme serveru obsluhovat pouze určitý, předem daný, počet klientů. Jestliže tedy dojde k požadavku o připojení klienta a přitom server již dosáhl maximálního počtu připojených klientů, server klienta odpojí.

Další možností jak omezit efektivitu tohoto útoku spočívá v určení limitu velikosti přesouvaných souborů. Server bude mít definováno, jak může být přesouvaný soubor maximálně veliký. Jestliže je tato hodnota překročena, vypíše se o tom hlášení a příkaz k přesunu dat se neprovede.

Zabezpečení z hlediska OS

Zajistit bezpečnost serveru musíme i z hlediska prostředí ve kterém je server spuštěn, tedy z hlediska operačního systému. Jelikož v systému, ve kterém je server spuštěn, může být nedostatek zdrojů pro zpracování požadavků, musí server tyto nedostatky brát v úvahu. Nemůže používat více prostředků než kolik může mít reálně k dispozici. Jedná se hlavně o velikost paměti a volné místo na disku.

Snaha o alokaci paměti musí být hlídána a při neúspěchu musí server zachovat svou funkčnost a nesmí zhavarovat. Při neúspěšné alokaci tedy vypíše hlášení o nedostatku prostředků a provádění příkazu, u kterého k tomu došlo, ukončí a pokračuje dále v příjmu dalších příkazů.

Nedostatek volného místa na disku je další důležitá oblast, na které může server zhavarovat. Každé operaci zápisu na disk, musí předcházet kontrola, zda na disku existuje dostatek volného místa. Při nedostatku místa se operace zápisu neprovede a vypíše se hlášení.

5.4 Klient

Klient je hlavní částí aplikace. Přes grafické rozhraní a pomocí komunikace se serverem zajišťuje interakci uživatele s rozpoznávačem slov. Pro komunikaci se serverem používá komunikační protokol popsaný v kapitole 5.2.

Návrh klienta popisuje způsob, jakým budou uživateli zobrazovány a interpretovány výsledky STK rozpoznávače zaslané ze serveru. Další funkcí klienta je načítání, zpracovávání a zobrazování zvukových souborů, které jsou vstupem STK rozpoznávače.

5.4.1 Zpracování zvuku

Aby jsme mohli v aplikaci pracovat se zvukem, musíme ho nejprve nějakým způsobem načíst. V počítačích je zvuk ukládán v souborech, které představují diskrétní interpretaci jeho signálu. Problémem je, že soubor obsahující tyto signály je značně obsáhlý, tudíž je vhodné ho nějakým způsobem komprimovat. Postupem času vzniklo tedy mnoho různých formátů, které se liší ve způsobu komprese.

Použitý formát zvukového souboru

V kapitole 4.3.2 jsme si určily, že není hlavním požadavkem schopnost číst co největší počet těchto formátů, ale postačí pouze nejběžnější a nejjednodušší formát WAV.

Každý zvuk uložený v podobě diskrétní posloupnosti čísel ve zvukového formátu obsahuje několik základních informací, které musíme znát, abychom mohli zobrazit jeho signál.

- **Frekvence vzorkování** (samplerate) značí počet vzorků (hodnot diskrétního signálu) za sekundu, čím větší hodnota, tím se zvuk více přibližuje reálu. Standardní hodnota je 44 000 Hz/s.
- **Počet kanálů** označuje zda jde o mono, stereo, či vícekanálový zvuk.
- **Počet bitů** na vzorek značí přesnost jednoho vzorku.
- **Počet vzorků** určuje délku signálu.

Čtení zvukového souboru

Čtení nekomprimovaného zvukového souboru je triviální. Pro zobrazování potřebujeme co nejrychlejší způsob čtení vzorků. Načteme tedy pomocí knihovny pro manipulaci se zvukovými soubory všechny vzorky do paměti počítače, tzn. celý zvukový soubor. To nám dovoluje rychlý přístup ke všem vzorkům najednou.

Zobrazení zvuku

Zobrazení zvuku je pak pouze pospojování jednotlivých vzorků signálu do křivky a vynesení do souřadného systému ve 2D rozměru.

Jelikož nemusí být vždy žádoucí zobrazit celý zvuk, musí mít klient definováno kolik vzorků, resp. jak velký časový úsek chce zobrazit a také, o kolik vzorků bude zobrazený signál posunut vůči počátku. Pomocí těchto proměnných pak vykreslujeme danou část signálu.

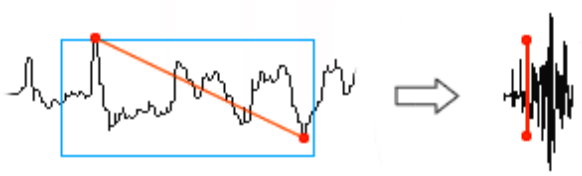
Funkcemi Zoom in a Zoom out si může uživatel zobrazený signál přibližovat a vzdalovat. Zobrazení signálu se pak provádí pomocí následujícího algoritmu:

```
x_inc = Šířka okna pro kreslení / Počet vzorků v okně
x_max = Min(Počet vzorků v okně, Celkový počet vzorků)
skip = Počet vzorků které přeskočit
for ( x = 0; x < x_max; x += skip )
{
    min = Minimální hodnota vzorku v intervalu x až (x + 1)
    max = Maximální hodnota vzorku v intervalu x až (x + 1)

    Kresli čáru(x0 * x_inc, min, x * x_inc, max)

    x0 = x
}
```

V algoritmu vidíme, že při zobrazení signálu dochází k přeskakování vždy stejně velkého bloku vzorků. Velikost skoku je určena proměnnou `skip`. Aby nedocházelo při posunutí signálu k nepravidelnostem v jeho zobrazení, používá se při výpočtu minimálního a maximálního vzorku tzv. obálka, která zajistí správné zobrazení.



Obrázek 5.4: Obálka signálu.

Při Zoom in a Zoom out se pouze upravuje hodnota proměnné `Počet vzorků na okno`. Funkce Zoom in má omezení při přibližování nastaveno tak, že počet vzorků na okno nemůže být nižší, než je šířka okna v pixelech.

Přehrávání zvuku

Uživateli musí být umožněno přehrát jak celý načtený zvuk, tak i část vztahující se k určitému rozpoznanému klíčovému slovu. Tato část, která má svůj začátek a konec, může být přehrána včetně svého okolí, tzv. kontextu. Velikost tohoto kontextu si volí uživatel sám.

5.4.2 Interpretace výstupu rozpoznávače

Výstup z rozpoznávače je soubor slov, či písmen, které byly v procesu rozpoznávání detekovány. Každé slovo má, kromě jednoznačné identifikace počáteční a koncové pozice vzorku v signálu, také jistou pravděpodobnost se kterou bylo slovo detekováno. Jakmile klient obdrží ze serveru tyto informace, musí je přehledně zobrazit uživateli.

Jelikož známe začátek a konec vzorku každého detekovaného slova, můžeme jej zobrazit v signálu zvuku. K zobrazení pravděpodobností slouží zvláštní prostor, popsaný v následující kapitole 5.4.3.

5.4.3 Grafické uživatelské prostředí

Uživatelské prostředí by mělo být jednoduché, přehledné a intuitivní. Zajišťuje interaktivitu uživatele s programem a umožňuje mu tím zasahovat do procesu rozpoznávání.

Návrh grafického uživatelské prostředí vycházel z koncepce vytvořit co nejjednodušší prostředí, bez zbytečných funkcí a informací, které by zatěžovali uživatele a znepřehlednili by používání programu. Musíme si v prvé řadě ujasnit, co všechno potřebujeme zobrazit:

- **Menu** či **toolbar**, pomocí kterého bude uživatel dávat příkazy k připojení na server, posílání dat a zpracování dat na serveru.
- **Panel s výběrem** možností ze seznamu operací, které podporuje STK toolkit.
- **Panel informací** zobrazující základní informace o načteném zvukovém souboru.
- **Panel klíčových slov** by měl umožnit zadávat, načítat a ukládat seznam klíčových slov.
- **Panel zvukového souboru a pravděpodobností rozpoznaných slov** je důležitou částí. Jeho hlavní funkce je zobrazení signálu zvuku a zvýraznění jednotlivých rozpoznaných slov podle jejich pozice výskytu ve zvuku. Tento panel bude také zobrazovat pravděpodobnosti jednotlivých slov a pomocí posuvníku bude filtrovat dobře a špatně rozpoznaná slova, jejichž výsledek se bude zobrazovat v textovém panelu. Panel bude také přibližovat, vzdalovat a posouvat zobrazený zvuk.
- **Panel textového výstupu** má ukazovat textový výstup STK toolkitu a zvýrazňovat uživatelem vyfiltrované slova.

Rozložení jednotlivých prvků na ploše aplikace musí být logicky uspořádáno. Každý ze zobrazovaných prvků musí mít své jednoznačné místo. Jelikož rozpoznávání klíčových slov má svůj daný postup, popsaný v kapitole 2, budou podle tohoto postupu jednotlivé prvky zobrazeny pod sebou, podle pravidla „první informace, poté vstupy a nakonec výstupy“.

Menu s příkazy tedy bude v nejhořejší části aplikace, výběr ze seznamu operací a panel informací budou umístěny pod ním. Protože klíčová slova a zvukový soubor jsou vstupem aplikace, bude následovat panel klíčových slov a panel zvukového souboru. Výstup STK toolkitu se bude zobrazovat v posledním panelu textového výstup.

5.5 Konfigurovatelnost

V kapitole 4.4 jsme si uvedli, ze aplikace má být co nejvíce konfigurovatelná. Tedy aplikace by měla mít možnost načítat a ukládat svou konfiguraci, která bude určovat jednotlivá nastavení programu. Konfiguraci lze ukládat různými způsoby, v závislosti na použitém operačním systému.

Ve Windows existuje centrální místo pro ukládání konfigurace všech aplikací, tzv. registry. Naproti tomu Linux podporuje ukládání konfigurace pouze do souborů.

Jelikož navrhovaná aplikace má být multiplatformní, tj. spustitelná na více operačních systémech, zvolil jsem pro ukládání konfigurace tzv. `INI` soubory, který je dostupný na systému Windows i Linux. Jde o soubory, jež mají předem definovanou syntaxi ukládání jednotlivých konfiguračních údajů. Typický `INI` soubor může vypadat takto:

```
[section1]
; some comment on section1
var1 = abc
var2 = 451

[section2]
; another comment
var1 = 123
var2 = dfg
```

Server i klient tedy budou mít u sebe konfigurační soubory, ze kterých budou číst jednotlivá nastavení, která budou pro server i klienta naprosto odlišné.

Konfigurace serveru

Server bude číst z konfigurace následující nastavení:

- **Nastavení komunikace** určuje port na kterém má být server spuštěn.
- **Cesta ke skriptům**, pomocí nichž se budou spouštět jednotlivé funkce STK toolkitu.
- **Nastavení pracovního adresáře**, kde si bude server ukládat dočasné soubory.

Konfigurace klienta

U klienta to bude zejména toto nastavení:

- **Nastavení komunikace** určuje adresu a port na které bude klient připojovat.
- **Parametry sigmoidu** s jehož pomocí lze nastavit citlivost zobrazení výsledku na pravděpodobnost výskytu klíčových slov.
- **Nastavení barev** jednotlivých ovládacích prvků.

Kapitola 6

Implementace

V předchozí kapitole byl představen návrh síťového interface k detektoru klíčových slov. Dalším krokem při vytváření projektu bylo zvolení vhodné technologie k implementaci. V kapitole 6.1 je popsán výběr a popis jednotlivých technologií, další kapitoly pak představují implementaci serveru a klienta.

6.1 Výběr a popis technologií

Základním požadavkem pro implementaci, jenž ovlivnil volbu technologie bylo, aby projekt byl multiplatformní. Volba spočívala ve výběru programovacího jazyka a vhodné multiplatformní knihovny. Pro výběr programovacího jazyka byly rozhodující tyto oblasti:

- **Rychlost** je důležitá, protože navrhovaný systém pracuje s velkým množstvím zvukových dat, které se snaží zobrazovat.
- **Multiplatformnost** musí být zajištěna tak, aby při přenosu kódu z jednoho operačního systému do druhého nemuseli v něm být prováděny žádné změny.
- **Jednoduchost** spočívá v možnosti produkovat jednoduchý a srozumitelný zdrojový kód.
- **Rozšířenost** programovacího jazyka nám dává možnost rychlejšího vývoje kvůli jeho velké skupině vývojářů.
- **Cena** určuje náklady na překlad zdrojových kódu. Aby navrhovaný systém nemusel být omezen na proprietální překladač, tj. aby si mohl program zkompilovat každý uživatel, musí být překladač zdarma.

Po zvážení těchto kritérií probíhal výběr mezi platformou Java a C++. Jelikož oba programovací jazyky jsou značně robustní a existují pro ně překladače jež jsou zdarma, výběr byl nakonec porovnáním možnosti vytváření multiplatformních aplikací, jejich rychlosti a jednoduchosti použití.

C++ není třeba představovat. Tento jazyk existuje na všech hlavních platformách. Je to jeden z nejpoužívanějších programovacích jazyků, hlavně pro rychlost vytvořeného kódu a rozšířenost. Pomocí různých knihoven lze produkovat i multiplatformní kód.

Java je programovací jazyk pocházející od firmy Sun Microsystems. Jedná se o objektově orientovaný jazyk vycházející z C++. Velkou výhodou je její hardwarová s softwarová nezávislost, neboť je překládaná do speciálního mezikódu, který je na konkrétním počítači interpretován. Vytvořenou aplikaci je možno spustit na kterémkoliv operačním systému. Naopak její nevýhodou je její značná hardwarová náročnost.

Protože navrhovaný systém by měl být co nejrychlejší a v ostatních požadovaných oblastech se rozdíly v těchto dvou jazycích stírají, zvolil jsem k implementaci jazyk C++.

6.1.1 Použité knihovny

Samotný jazyk C++ má pouze standardní knihovny pro výpočty a základní IO operace. Aby mohl implementovaný systém používat technologii soketů, podporovat přehrávání a načítání zvuku, musí používat další knihovny. Jelikož v jazyce C++ není standardně zajištěna přenositelnost kódu mezi platformami, použité knihovny musí být multiplatformí.

Kvůli přenositelnosti kódu je pro základní zajištění technologie soketů a tvorbu grafického uživatelského prostředí použita knihovna wxWidgets. Dále pomocí knihovny libsndfile a PortAudio můžeme načítat a přehrávat zvukové soubory.

wxWidgets

WxWidgets [1] je otevřená knihovna poskytující programátorovi pohodlí při psaní multiplatformních aplikací. Hlavní podporované platformy jsou Unix a Windows. Její výhoda je v tom, že se nezaměřuje pouze na GUI, ale také na implementaci síťové a meziprocesorové komunikace, spuštění procesů, vlákna a načítání a ukládání konfigurace. Knihovna je implementována v C++, ale její používání je možné v mnoha běžně používaných programovacích jazycích. Podporuje široké množství kompilátorů, můžeme použít GCC nebo komerční kompilátor dodávaný se systémem, pod Windows pak libovolný z rozšířených kompilátorů (velmi dobře jsou podporovány Visual Studio C++, Mingw/GCC, Borland C++).

wxWidgets [4] je nejlépe popsán jako nativní toolkit. Místo napodobování grafiky prvků používá nativní grafické prvky na podporovaných platformách. Díky tomu je vzhled a chování výsledné aplikace na všech platformách stejné, jako při vývoji aplikace pouze pro danou platformu.

Důležitou součástí v návrhu GUI pod wxWidgets jsou tzv. sizery, které umožňují vytvářet dialogová okna bez toho, že byste zadávali souřadnice ovládacích prvků. Místo toho stačí popsat logickou strukturu dialogu vložením grafických prvků do jakýchsi boxů a o zbytek se postará knihovna. Výhodou je, že takto vytvořený dialog bude vypadat hezky na všech platformách a programátor se nemusí starat o to, jak je který prvek velký na jaké platformě.

V aplikaci byla použita verze knihovny 2.8.3.

Libsndfile

Libsndfile [2] je multiplatformní opensource knihovna pro čtení a tvorbu zvukových souborů. Jejími hlavními přednostmi je možnost čtení a zápisu velkého množství zvukových formátů. Je napsána v jazyce C.

Podporovanými formáty jsou `WAV PCM`, `WAV u-law`, `WAV A-law`, `AIFF`, `AU` a `FLAC`.

[1] `http://www.wxwidgets.org/`
[2] `http://www.mega-nerd.com/libsndfile/`

V aplikaci byla použita verze knihovny 1.0.17.

PortAudio

PortAudio [3] je multiplatformní opensource knihovna pro přehrávání a nahrávání zvuku. Dovoluje programátorovi psát programy pro práci se zvukem s jednoduchým přístupem ke zvukovému zařízení. Hlavní předností oproti obdobným knihovnám je přenositelnost.

PortAudio je velmi jednoduchá API používající callback funkci, s jejíž pomocí lze na pozadí běhu aplikace přehrávat, či zpracovávat zvuk.

V aplikaci byla použita verze knihovny V19.

6.2 Server

Server je naprogramován jako konzolová aplikace. Implementace serveru je složena ze dvou tříd. `Serverapp` zajišťuje navazování spojení s klienty a vytváření vláken třídy `MyThread` obsluhující požadavky jednotlivých klientů.

Třída `serverapp` je hlavní třídou aplikace. Při inicializaci nastaví logování aplikačních výstupu, načte nastavení z konfiguračního souboru a spustí naslouchání na nastaveném portu.

Jelikož naslouchání je neblokující příkaz, musíme zamezit plnému využití CPU serverem a přenechat tak výpočetní kapacitu ostatním aplikacím. Je tudíž nutné v pravidelných intervalech proces naslouchání na krátký čas přerušit. To provedeme pomocí příkazu `wxThread::Sleep(10)`, jenž aplikaci uspí na 10 milisekund, čímž dojde k uvolnění prostředků CPU dalším aplikacím v operačním systému. To stejné musíme provést i v každém vlákně.

Veřejná část (Public)	
`OnInit`	Inicializace serveru
`OnRun`	Navazuje spojení při připojení klienta
`OnExit`	Dealokuje vytvořený soket a ukončí server
Soukromá část (Private)	
`ReadConf`	Čte konfigurační soubor
`WriteDefaultConf`	Vytváří přednastavený konfigurační soubor

Tabulka 6.1: Metody třídy `serverapp`.

Jakmile na nastaveném portu přijde požadavek klienta o připojení, server vytvoří nové vlákno s nízkou prioritou a předá mu připojený soket. Při tom se předa také řetězec obsahující prefix pro ukládání souborů přijmutých ze serveru.

Vytvořené vlákno nejprve nastaví parametry soketu. Jedná se především o příznak blokujícího spojení a nastavení timeoutu. Následuje odeslání nastavení STK toolkitu pomocí metody `SendServerCFG`. Po spuštění vlákna příkazem Entry začne server na daném soketu přijímat příkazy klienta definované v kapitole 5.2.

Každý z příkazů má definovanou ekvivaletní metodu, jenž má na starost jejich obsluhu. Metoda `ReceiveWAV` přijímá `wav` soubor, `ReceiveKWS` přijímá soubor klíčových slov, `ExecuteSTK` spouští rozpoznávání na příjmutých souborech a `SendMLF` odesílá klientovi výstup rozpoznávače. Další metoda `SendClientCFG` odesílá klientovi jeho přednastavenou konfiguraci.

[3] `http://http://www.portaudio.com/`

Obsluha jednotlivých komunikačních příkazů je uvedena na následujícím kódu:

```
do {
    if (socket->IsData())
    {
        socket->ReadMsg(&op, sizeof(op));
        switch (op)
        {
            case OP_WAV:
                ReceiveWAV();
                break;

            case OP_KWS:
                ReceiveKWS();
                break;

            case OP_PROCESS:
                if (ExecuteSTK())
                  SendMLF();
                break;
        }
        op = -1;
    }

    if (socket->Error() || socket->LastError() != 0)
        break;

    Sleep(100);
    wxThread::Yield();
} while (true);
```

Samotný přenos dat zajišťují funkce `ReadSocketToFile` pro příjem souborů, `WriteFileToSocket` pro odeslání souborů a `WriteStringToSocket` pro odeslání řetězce. Jejich implementace spočívá v odeslání či příjmu čísla udávající velikost přenášených dat, následované přenosem samotných dat.

Po ukončení obsluhy server soket uzavře a vymaže dočasné soubory, které se vytvořili po dobu trvání spojení.

Veřejná část (Public)	
`MyThread`	Inicializace vlákna
`Entry`	Spuštení vlákna
Soukromá část (Private)	
`ReceiveWAV`	Přijímá od klienta zvukový soubor WAV
`ReceiveKWS`	Přijímá od klienta soubor klíčových slov
`ExecuteSTK`	Spouští STK toolkit na přijmutých souborech
`SendMLF`	Výstup z STK toolkitu posílá zpět klientovi
`SendServerCFG`	Posílá klientovi nastavení možností STK toolkitu
`SendClientCFG`	Posílá klientovi přednastavenou konfiguraci
`ClearAll`	Vymaže přijmuté soubory a výstup STK toolkitu

Tabulka 6.2: Metody třídy `MyThread`.

Zabezpečení serveru bylo implementováno podle návrhu definovaném v kapitole 5.3.3. Lze definovat maximální velikost přenášeného souboru a maximální počet najednou připojených klientů. Tyto hodnoty lze nastavit v konfiguračním souboru. Jeho obsah je uveden v příloze C.

6.3 Klient

Implementace klienta je rozdělena na dva druhy tříd, jeden obstarávající nízkoúrovňové funkce a druhý obsahující třídy grafického uživatelského prostředí.

6.3.1 Třídy nízkoúrovňových funkcí

Ty mají na starost načítání konfigurace, práci se zvukovými soubory a se soubory výstupu rozpoznávače. Jsou implementovány třídy `config`, `wave` a `keywords`.

Třída config

Třída config je potomkem třídy `wxFileConfig` z knihovny wxWidgets. Pomocí metody `ReadAll` načítá konfiguraci z INI souboru a proměnné `g_CommonPrefs` přiřazuje jednotlivé její hodnoty. Tato proměnná je struktura obsahující nastavení jednotlivých částí klienta. Aby jsme zajistili přístup k načtené konfiguraci z celého programu, musí být tato proměnná globální.

Ekvivalentně metoda `WriteAll` zapisuje konfiguraci z proměnné `g_CommonPrefs` zpět na disk. Jestliže na disku chybí konfigurační soubor, použije se standardní konfigurace definovaná v programu.

Třída wave

Pomocí knihovny libsndfile může tato třída načítat zvukové soubory do paměti. Při inicializaci této třídy se pouze nastaví jméno zvukového souboru a až při volání hlavní metody `PrepareSamples` dojde k načtení souboru, výpočtu maximální hodnoty vzorku v souboru a definování počtu zobrazených sekund. Pomocí těchto hodnot pak zobrazujeme signál.

Třída také obsahuje metody pro posunu v signálu, `SetOffset` a `GetOffset`. Jelikož klient umožňuje i funkci přibližování a vzdalování zobrazeného signálu, definujeme proto funkce `SetZoom` a `GetZoom`.

Pro výpočet obálky signálu v daném intervalu zde existuje funkce `GetSampleEnvelope`.

Třída keywords

Tato třída obsahuje seznam jednotlivých klíčových slov. Každé klíčové slovo má strukturu danou počátečním a koncovým časem výskytu v signálu, klíčovým slovem a jeho pravděpodobností výskytu.

```
typedef struct {
  double start;
  double end;
  wxString word;
  double treshold;
} t_keyword;
```

Třída keywords definuje metody pro práci s klíčovými slovy. Metoda `AddKeyword` a `GetKeyword` provádí manipulaci s jednotlivými položkami v seznamu a pomocí metody `ParseMlf` načítáme do seznamu všechny klíčová slova v podobě jaké vyprodukoval STK toolkit na svém výstupu, tj. ve formátu MLF. Toto čtení pobíhá pomocí regulárních výrazů.

Aby mohl uživatel vybrat jednotlivá klíčová slova, jsou ve třídě implementovány metody `SetSelected` a `GetSelected`.

6.3.2 Třídy a prvky grafického uživatelského prostředí

Grafické prostředí je rozděleno na několik samostatných částí, kde každá část plní jednu z funkcí návrhu klienta:

- Hlavní toolbar, neboli obdoba hlavního menu.
- Panel operací
- Panel informací
- Panel klíčových slov
- Panel zvukového souboru.
- Panel textového výstupu.

Třída clientFrame

Je hlavní třídou klienta. Spojuje jednotlivé části GUI do jednoho celku a vytváří tak „most“ mezi uživatelem a funkcemi jednotlivých tříd. Tato třída také obstarává obsluhu událostí a zajišťuje připojení a komunikaci se serverem.

Metody `Connect` a `Disconnect` navazují/ruší spojení klienta se vzdáleným serverem. Jakmile klient naváže spojení, komunikuje se serverem pomocí metod `SendWav`, `SendKws` a `Process`. Jejich význam je obdobný jako u metod v implementaci serveru. Krom toho je zde navíc metoda `GetCfg`, která dává serveru příkaz pro odeslání přednastavené konfigurace klienta ze serveru.

Následuje popis jednotlivých části grafického uživatelského prostředí.

Hlavní toolbar

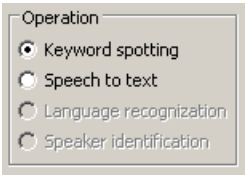
Pomocí hlavního toolbaru zadává uživatel základní příkazy aplikace. Jedná se především o připojení k serveru, načtení zvukového souboru, poslání příkazu k provedení rozpoznávání a požadavku na získání přednastavené konfigurace klienta.

Obrázek 6.1: Hlavní toolbar.

Panel operací

V panelu operací vybíráme mód rozpoznávání. STK toolkit podporuje detekci klíčových slov, převod řeči na text, rozpoznávání mluvčího a rozpoznávání jazyka.

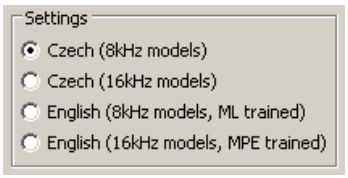
Po připojení k serveru si klient stáhne podporované módy rozpoznávání a podle nich nastaví možnosti výběru v panelu operací.



Obrázek 6.2: Panel operací.

Panel nastavení STK toolkitu

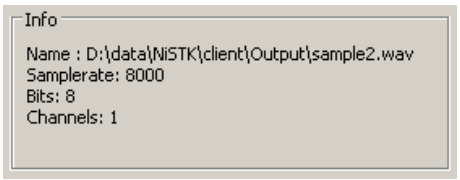
Panel nastavení je pojat obecně. Na serveru je definováno několik konfigurací STK toolkitu. Může jít o konfiguraci pro různé jazyky, modely postavené na různé frekvenci samplování, či různě natrénované neuronové sítě. Pomocí tohoto panelu můžeme přepínat mezi jednotlivými konfiguracemi.



Obrázek 6.3: Panel nastavení STK toolkitu.

Panel informací

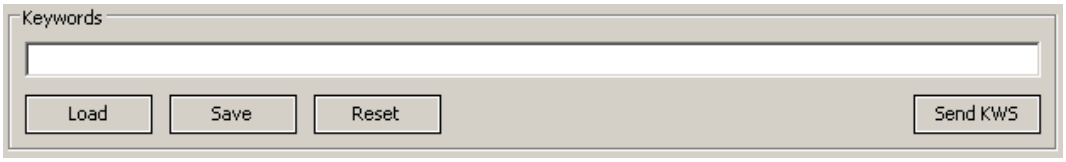
Obsahuje pouze vlastnosti načteného zvukového souboru, tj. jeho název, frekvenci vzorkování, počet bitů na vzorek a počet kanálů.



Obrázek 6.4: Panel informací.

Panel klíčových slov

V panelu klíčových slov může uživatel zadávat jednotlivá klíčová slova. Je implementováno také jejich ukládání a načítání ze souboru. Při stisku tlačítka „Send KWS“ se odešle na server pouze seznam klíčových slov.



Obrázek 6.5: Panel klíčových slov.

Panel zvukového souboru

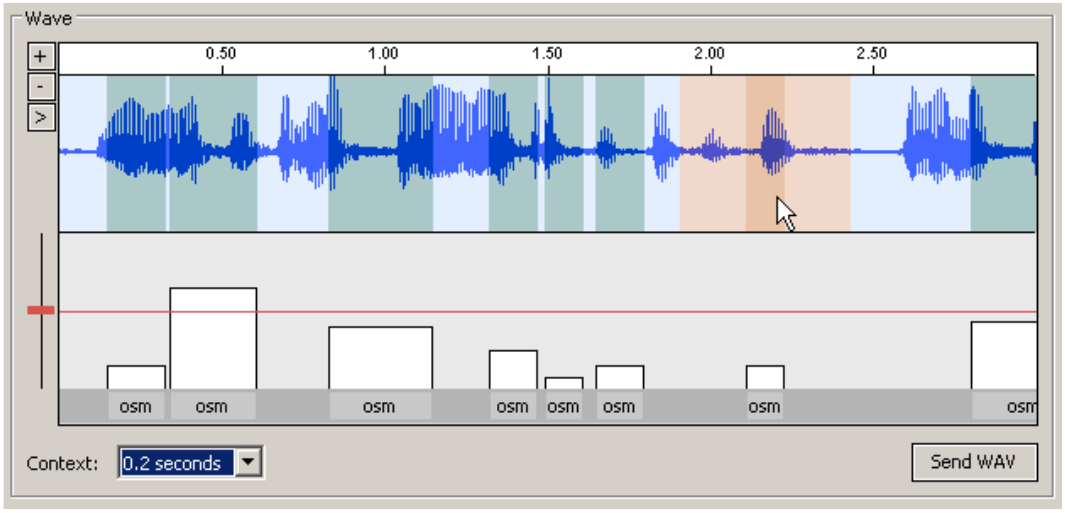
Je implementován ve třídě `trackPanel`, jenž obsahuje hlavně metody pro zobrazení signálu, časového měřítka a pravděpodobností klíčových slov. Algoritmus pro vykreslování signálu byl popsán v kapitole 5.4.1.

Panel má i tlačítka pro přibližování, vzdalování a přehrávání zobrazeného signálu. Po příjmutí výsledku rozpoznávání ze serveru se zobrazí jednotlivá klíčová slova a pomocí sloupců se graficky znázorní jejich pravděpodobnost. Výška sloupce se vypočítá pomocí funkce sigmoidu 6.1, kde A a B jsou předem definované konstanty, uložené v konfiguračním souboru.

$$P(t) = \frac{1}{1 + e^{(A - B*t)}} \tag{6.1}$$

Pro filtrování detekovaných slov slouží horizontální posuvník, jehož hodnota je v podobě čáry promítána do zobrazení pravděpodobností. Jakmile je sloupec (pravděpodobnost slova) nad touto čárou, je slovo vyhodnoceno jako správně detekované a zvýrazní se v panelu textového výstupu.

Lze přehrávat také jednotlivá klíčová slova včetně jejich kontextu (okolí). Velikost kontextu lze nastavovat v „selectboxu“ ve spodní části panelu. Pomocí tlačítka „Send WAV“ odesíláme na server pouze nahraný zvukový soubor.



Obrázek 6.6: Panel zvukového souboru.

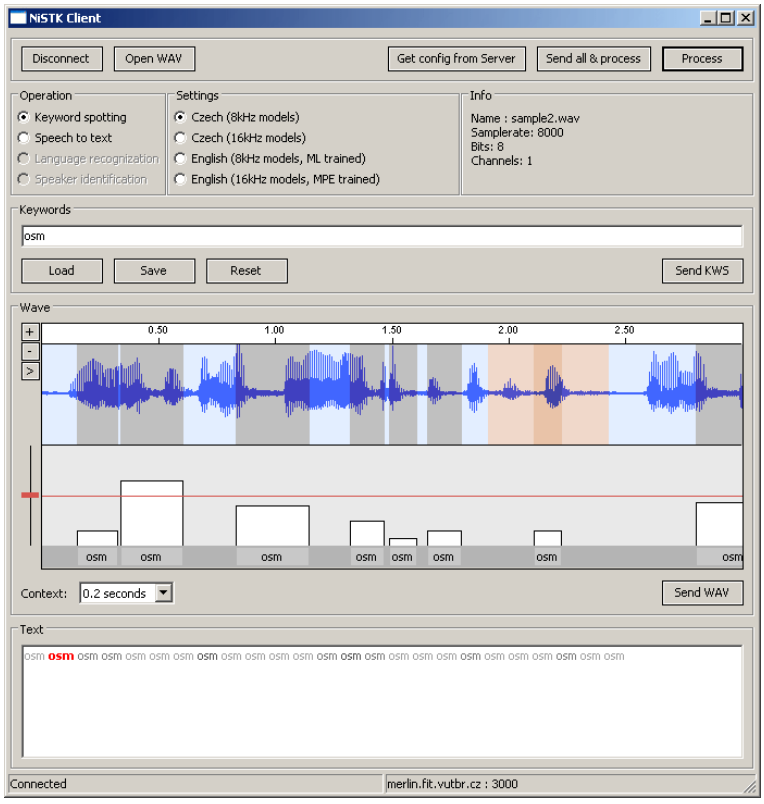
Panel textového výstupu

Po rozpoznání zvuku na serveru a přijmu výsledků zobrazí tento panel v řadě za sebou jednotlivá rozpoznaná slova. Podle výše pravděpodobnosti výskytu se každé slovo zvýrazní různě silnou barvou šedi.

Obrázek 6.7: Panel textového výstupu.

6.3.3 Grafické uživatelské prostředí

Implementovaná aplikace klienta je tedy implementována a její výslednou podobu pod operačním systémem Windows můžeme vidět na obrázku 6.8. Aplikace je multiplatformní a lze ji tedy spustit i v dalších operačních systémech. Vzhled aplikace pod OS Linux je v příloze A.



Obrázek 6.8: Grafické uživatelské prostředí klienta.

Kapitola 7

Rozšíření systému

Navržená a implementovaná aplikace dokáže pracovat s STK toolkitem pouze v offline módu, neboť STK toolkit neumí online rozpoznávání a v době implementace nebylo známo ani jeho rozhraní. Tento fakt je určující pro směr dalšího vývoje. Následující řádky proto popisují možnosti rozšíření.

7.1 Online mód

Online mód pracuje s rozpoznávačem v reálném čase tak, že klient nahráva z mikrofonu zvuk a posílá ho v reálném čase serveru. Ten přijmuté data předá ihned rozpoznávači ke zpracování. S malým zpožděním pak server obdrží rozpoznaná data a pošle je zpět klientovi. Toto schéma komunikace lze úspěšně řešit dvěma způsoby.

7.1.1 Rozšíření komunikačního protokolu

Jednodušším a možná i elegantnějším řešením je rozšíření stávajícího komunikačního protokolu o další příkaz sloužící k přenosu malého konstatního bloku zvukových dat. Nahrávané data by si klient postupně ukládat do fronty a pomocí tohoto příkazu by se je snažil odesílat. Server by pak pomocí API knihovny k online rozpoznávání inicioval volání rozpoznávače na přijmutem bloku dat.

Zaslání výsledků vrácených rozpoznávačem zpět klientovi by zajišťoval další příkaz. To proto, aby klient novými výsledky nepřepsal ty staré, ale aby je připojil k již stávajícím výsledkům.

7.1.2 Komunikace na protokolu UDP

Protokol UDP je nespolehlivý nespojovaný protokol a tudíž nezaručuje, zda přenášený paket neztratí, nezmění pořadí paketů, nebo zda se některý paket nedoručí vícekrát. Díky tomu je UDP rychlejší a efektivnější a je tedy vhodný na streaming dat.

Klient by tedy navázal spojení se serverem jak přes TCP protokol, tak i přes UDP běžícím na jiném portu. Komunikace by pak spočívala v odesílání zvukových dat přes UDP a příjem výsledků přes TCP protokol. Problém tohoto přístupu by byl v možné ztrátě či duplicitě dat zvukových dat, vyplývající z podstaty UDP protokolu.

Kapitola 8

Závěr

Tato práce zpracovává téma z oblasti síťové komunikace a rozpoznávání řeči. Protože se jedná o velmi rozsáhlou oblast, věnuje se druhá kapitola oblastem rozpoznávání řeči a detekci klíčových slov. Třetí kapitola pak popisuje význam síťových protokolů a obecný způsob komunikace mezi dvěma účastníky v síti.

Po analýze požadavků ve čtvrté kapitole, zpracovává pátá kapitola návrh systému, který definuje komunikační protokol na architektuře klient/server, pravidla pro práci serveru s STK toolkitem, bezpečnost a návrh grafického uživatelského rozhraní klienta. Tato část diplomové práce byla pro výslednou podobu systému rozhodující. Navržené uživatelské prostředí je jednoduché, přehledné a intuitivní. Vychází z koncepce vytvořit co nejjednodušší prostředí, bez zbytečných funkcí a informací, které by zatěžovali uživatele.

Pro vývoj aplikace, popsaný v šesté kapitole, byla vybrána knihovna wxWidgets, jejíž hlavní výhoda spočívá v možnosti produkovat multiplatformní aplikace. Navržený komunikační protokol jsem implementoval, včetně opatření potřebných pro zabezpečení aplikace. Implementovaný systém je je připraven na další rozšiřování. Následný vývoj by se měl zaměřit na implementaci online rozpoznávání, popřípadě na další rožšíření funkčnosti.

Implementace serveru, klienta a jeho grafického uživatelského prostředí vytváří síťový interface k detektoru klíčových slov.

Hlavním přínosem je v možnosti použit STK toolkit jako klient/server aplikaci, distribuovat funkce STK toolkitu přes síť více uživateli najednou a přehledně zobrazovat výstupy rozpoznávání.

Díky této diplomové práci jsem se seznámil s oblastí rozpoznávání řeči, prohloubil si znalost problematiky sítí a získal zkušenosti při návrhu multiplatformního grafického uživatelského prostředí.

Literatura

[1] WWW stránky. The htkbook. `http://nesl.ee.ucla.edu/projects/ibadge/docs/ASR/htk/htkbook.pdf`. (březen 2007).

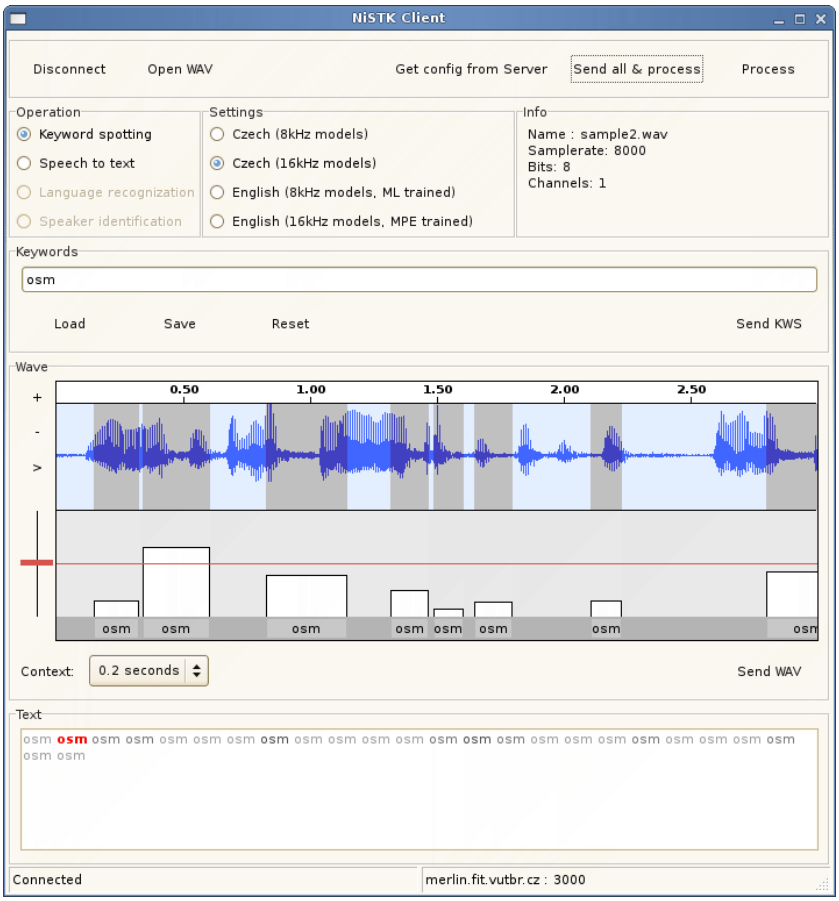
[2] Libor Dostálek Alena Kabelová. *Velký průvodce protokoly TCP/IP a systémem DNS*. Computer Press, 2002. ISBN 80-7226-675-6.

[3] WWW stránky. Beej's guide to network programming. `http://beej.us/guide/bgnet`. (leden 2007).

[4] Stefan Csomor Julian Smart, Kevin Hock. *Cross-Platform GUI Programming with wxWidgets*. Prentice Hall PTR, 2005. ISBN 0131473816.

Dodatek A

Grafické uživatelské prostředí pod OS Linux



Obrázek A.1: Vzhled grafického uživatelského prostředí pod operačním systémem Linux.

Dodatek B

Skript pro spuštění STK toolkitu

```
#!/bin/bash

#kws.sh input.wav keywords.txt output.mlf

inputwavfile=$1
inputkws=$2
outputmlf=$3
mode=$4

inputrawfile=${inputwavfile%.wav}.raw

sox -t .wav ${inputwavfile} -t .raw -r 8000 -s -w -c 1 ${inputrawfile}
    resample

cat lexicon | grep -f ${inputkws} > keywords.dict

create_CI_net_for_KWS_new.pl \
  phonemes \
  keywords.dict \
  phnrec/phn_cz_spdat_lcrc_n1500_kws/net/network


phnrec/phnrec \
  -c phnrec/phn_cz_spdat_lcrc_n1500_kws \
  -i ${inputrawfile} \
  -o ${outputmlf}

exit 0
```

Skript pro detekci klíčových slov

Dodatek C

Konfigurační soubory

```
[General]
port=3000
tmpdir=./tmp
transfer_timeout=5
cleaning=0
log=1
log_file=debug.log
client_config_file=client.cfg
kws_script=kws.sh
stt_script=stt.sh
lid_script=
spk_script=
params=Czech (8kHz models);English (16kHz models, ML trained)
```

Konfigurační soubor serveru.

```
[General]
port=3000
host=merlin.fit.vutbr.cz
log=1
transfer_timeout=15
sigmoid_a=-2
sigmoid_b=0.05

[Colors]
wave_pen=#5577FF
wave_bg=#E3EEFF
wave_select=#E9C3A8
wave_select_context=#F0D8CA
wave_keyword=#C0C0C0
treshold_pen=#000000
treshold_brush=#FFFFFF
treshold_bg=#E9E9E9
```

Konfigurační soubor klienta.