Vstup zvuku a spektrální analýza

Petr Schwarz
(schwarzp@fit.vutbr.cz)

Stáhněte si prosím, rozbalte, zkompilujte a spusťte:

www.fit.vutbr.cz/~cernocky/speech/labs/03_c*ř*ec .tgz

Motivace

Pro vybudování systému pro rozpoznávání řeči jsou zapotřebí znalosti ze spousty oblastí:

- zpracování signálu
- programovacího jazyka (C/C++)
- paralelního programování
- akustického modelování
- jazykového modelování
- prohledávání graf
- zobrazování (jak kdy)

Našim cílem je postupně Vás se vším seznámit.

Co dnes?

1) Stručný úvod do C++
2) On-line vstup zvuku pod Linuxem
3) Popis tříd jenž dostanete k dispozici

   (třídy: on-line vstup zvuku pro Linux a Windows, dělení na rámce, vykreslování grafu pomocí Curses)

4) Vykreslovaní energie signálu a spektrograf – ukázka toho, jak vše použít
5) Projekt

Základy C++

(Ukážeme si ve stručnosti jen to, co použijeme,
nebo s čím bychom se mohli setkat)

Alokace paměti

Jazyk C

```
char *mem;
mem = malloc(n);
mem = calloc(n, size);
…
free(mem);
```

C++

```
char *mem;
mem = new char[n];
…
delete [] mem;
```

Třídy

- Obsahují data (proměnné) a funkce (metody)
- Třída je jen“ vzor”, “typ”, “předpis”, sama o sobě neexistuje!

Trida.h

```
class A
{
    protected:
        int promena;
    public:
        A();
        ~A();
        int funkce();
};
```

Trida.cpp

```
#include "Trida.h

A::A()     // konstruktor
{ }

A::~A()    // destruktor
{ }

int A::funkce()
{ return 0; }
```

Instance třídy

- Skutečná „fyzická“ realizace. Skalární proměnné se vytvoří, dynamicky alokované proměnné by se měly alokovat v konstruktoru (ten je vyvolán po vytvoření instance)
- Vytvoření instance:

Staticky:

```
A a;
A a(10, 2)    // pokud má konstruktor parametry
```

Dynamicky:

```
A *a;
a = new A;
a = new A(10, 2);
…
delete A
```

```
A *a;
a = new A[10];   // 10 instanci
…
delete [] a;
```

Dědičnost

- Definice nové třídy na základě staré, většinu věcí chceme nechat, ale něco přidat a u něčeho změnit fungování

```
class A
{
    protected:
        int promena;
    public:
        A();
        ~A();
        int funkce();
};
```

```
class B : public A
{
    public:
        B();
        ~B();
        int nova_funkce();
};
```

Virtuální funkce

- Chceme změnit chování některé funkce ze třídy A
- Funkce ‘funkce’je p ekryta funkcí se stejným jménem ze třídy B

```
class A
{
        virtual int funkce(a, b) {return a + b;};
};

class B : public A
{
        int funkce(a, b) {return a - b;};
};
```

Statické funkce

- Občas je potřeba předat ukazatel na některou funkci třídy aplikačnímu rozhraní systému. Ten ji pak volá při určité události, například skončení nahrávání jednoho rámce řeči.
- Ukazatel na funkci třídy se skládá ze dvou částí, ukazatele na funkci a ukazatele na data (vlastní třídu), proto jej nelze předat přímo.

Statické funkce II

```
class A
{
    static void CALLBACK func(HWAVEIN hwi, UINT MSG,
      DWORD dwInstance, DWORD dwParam1, DWORD dwParam2)
    {
        A *a = (A *)dwInstance;
        if(uMSG == WIM_DATA) a->onData();
    }
    void OnData()
    { … }                  // Zpracovani dat
};

waveInOpen(&device, WAVE_MAPPER, &wf, (DWORD)func,
   (DWORD)this, CALLBACK_FUNCTION))
```

Klíčové slovo volatile

- Kompilátor zajistí, že kód kolem této proměnné nebude optimalizován a proměnná bude určitě v paměti (jinak by ji mohl „naoptimalizovat“ do registru)
- Proč? Protože budeme chtít přistupovat ze dvou paralelně běžících částí kódu (vláken)

```
volatile promena;
```

bez volatile

```
A -> REG
REG = REG + 1
…
REG = REG * 10
```

s volatile

```
A -> REG
REG = REG + 1
REG -> A
…
A -> REG
REG = REG * 10
REG -> A
```

Výjimky

- Efektivní způsob obsluhy chyb

```
try
{
   …
   if(chyba) throw “tady je chyba”;
   …
}
catch(char *err)
{
   fprintf(stderr, “ERROR: %s\n”, err);
   exit(1);
}
```

Šablony

- Umožňuje vytvořit část kódu, který lze použít vícekrát s jiným typem

```
Matrix<float> A(10, 10);
Matrix<int> B(10, 10);
```

- Existuje STL (Standard Template Library), která je zdrojem velkého množství algoritmů, např. seznamy, řetězce, hashe, výjimky atd. Je dostupná v GCC, Visual C++ i CBuilderu

```
www.sgi.com/tech/stl
```

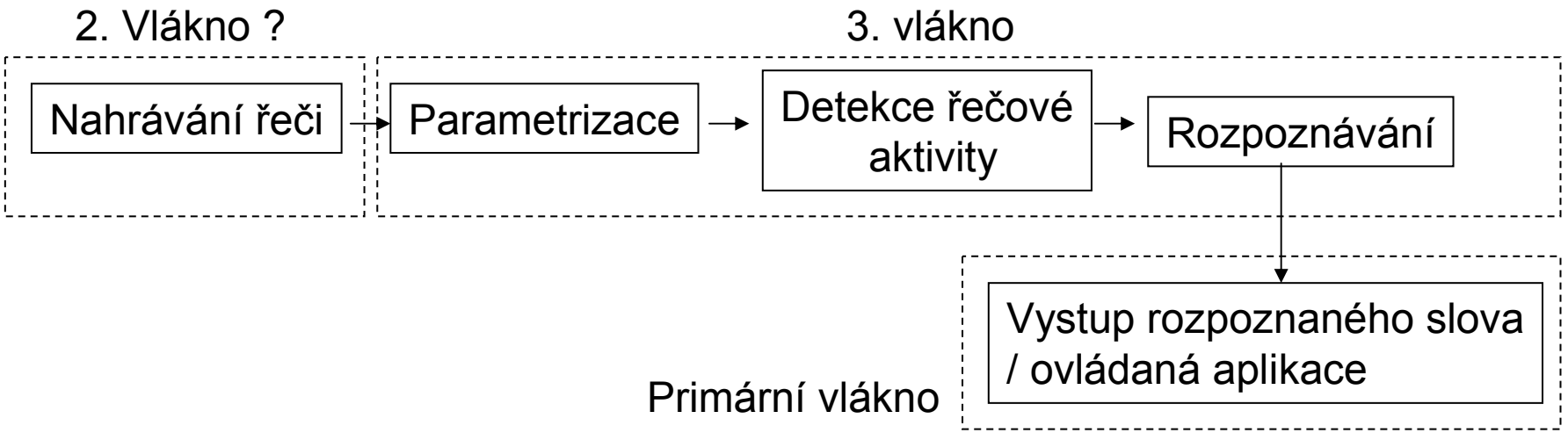
To byzatím k C++ stačilo… ☺

Struktura rozpoznávače

• Je potřeba zajistit, aby nahrávání jelo nepřetržitě, výpočet se nestihne mezi dvěmi vzorky (buffery) načítanými ze zvukové karty

• Procesor potřebujeme i pro jiné operace programu (vstup z jiných zařízení, výstup)

=> proto vlákna



Vstup zvuku

- Třída LWFSource, soubory lwfsource.[h, cpp] pro Linux a WFSource pro Windows (wfsource.[h, cpp])
- Zvukovka v Linuxu:

  - tváří se jako speciální soubor:

    ```
    cat /dev/dsp > soubor.raw
    ```

    …podívat se pomocí ws (WaveSurfer)

  - nastavení parametrů pomocí funkce ioctl, všechny možné parametry jsou v hlavičkovém souboru soundcard.h

  - čtení pomocí funkce read

    ```
    read(fileno(dp), storing, frameLen);
    ```

  - více: www.opensound.com/linux.html

Poznámky pro vstup zvuku pro Windows

- Pod Windows používáme knihovnu mmsystem.dll (Microsoft Multimedia System)
- K programu přilinkováváme zástupnou knihovnu winmm.lib

```
#pragma comment(lib, "winmm.lib")
```

- Pod Visual C++ tato knihovna existuje, pod CBulderem ji musíme vytvořit:

```
implib winmm.lib mmsystem.dll
```

Čtení vzorků ze zvukovky jede ve vlastním vláknu

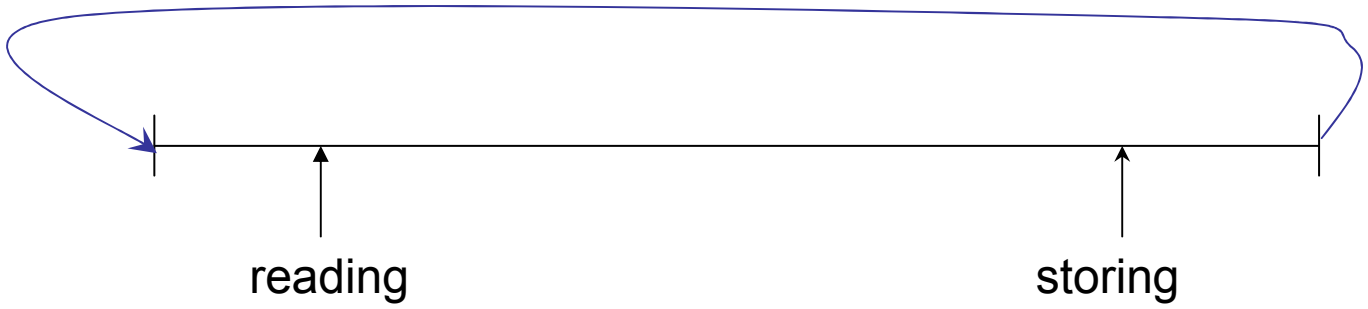
- Pro Linux knihovna phtread

  www.llnl.gov/computing/tutorials/workshop/workshop/pthreads/MAIN.html
- Linkovaní

  g++ program.cpp –lpthread

Čtení vzorků ze zvukovky a ukládání do bufferu, čtení z programu

- kruhový buffer:



storing – buffřík o velikosti FrameLen

reading – ukazatel odkud čteme

bytesRecorded – počet volných (nepřečtených) vzorků ve velkém bufferu

Založení vlákna

```
class A
{
    static void *recordingBypass(void *arg)  { … ; pthread_exit(0); };
};


pthread_t thread;
pthread_attr_t attr;

pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

pthread_create(&thread, &attr, recordingBypass, this)
pthread_attr_destroy(&attr);

…

int status;
pthread_join(thread, (void **)&status);
```

Synchronizace vláken

- Z bufferu, do kterého právě zapisujeme ze zvukovky se ve stejném okamžiku nesmí číst. Kritický není vlastní zápis, ale

  1) Posouvání mezí při dokončení zápisu
  2) Čtení

```
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);
```

```
    //      vlakno 1.
pthread_mutex_lock(&mutex);
…  // posunutí mezí po zápisu
pthread_mutex_unlock(&mutex);
```

```
        //  vlakno 2.
pthread_mutex_lock(&mutex);
…   //  čtení
pthread_mutex_unlock(&mutex);
```

```
pthread_mutex_destroy(&mutex);
```

Čekání vlákna

- Pokud nejsou k dispozici vzorky ze zvukovky, je třeba zajistit inteligentní čekání na hlavní vlákno

```
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);
pthread_cond_t cond;
pthread_cond_init(&cond, NULL);
```

```
//      zápis
pthread_mutex_lock (&mutex);
pthread_cond_signal(&cond);
pthread_mutex_unlock (&mutex);
```

```
//  čtení
pthread_mutex_lock (&mutex);
pthread_cond_wait(&cond, &mutex);
pthread_mutex_unlock (&mutex);
```

```
pthread_mutex_destroy(&mutex);
pthread_cond_destroy(&cond);
```

Architektura systému - plovoucí energie nebo spektrograf

Interface třídy pro nahrávání zvuku

1) LWFSource – vstup zvuku z mikrofonu a buffrovaní

```
class LWFSource
{
    public:
      LWFSource();
      ~LWFSource();
      bool isAvailable(char* ret_whay_not = 0);
      bool isOpen() {return dp != 0;};
      void open();
      void close();
      void read(char *buff, int n);
      void setFormat(int sf, int ch, int bps)
      void setDevice(char *d) {device = d;};
};
```

Interface třídy pro tvorbu rámců

2) Frames - Tvorba rámců s překrytím, parametry ve frames.h

```
class Frames
{
   protected:
       virtual void processFrame(float *frame, float
                          *ret_features);
   public:
                Frames();
               ~Frames();
                void reset();
                void addWaveform(float *waveform, int len);
                int getFeatures(float *ret_features);
};
```

Vykreslování grafu

- SpecGraf - Vykreslení grafu v textovém módu
- Používá knihovnu curses
- Linkovaní: g++ program.cpp -curses

```
class SpecGraf
{
  public:
      SpecGraf();
      ~SpecGraf();
      void set(int y, int x, int dy, int dx,
            int nbins, float max_val,
            float min_val);
      void show(float *data);
};
```

Plovoucí energie

- Honzova třída Eng, způsob odvození od Frames, propojení tříd, main
- viz zdrojáky!

Vaše práce

- Vytvořit třídu Spect odvozením od třídy Frames a dopsat spektrální analýzu

- Projekt:

  1) předělat program na zobrazování spektrografu pouze v určitém rozsahu frekvencí a v určitém počtu pásem.

  2) volitelně grafické rozhraní, plovoucí spektrogram (v úrovních šedi nebo barevný), portování pod Windows

Konec