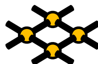
Vláknové programování
část XII

Lukáš Hejmánek, Petr Holub

`{xhejtman,hopet}@ics.muni.cz`



Laboratoř pokročilých síťových technologií

PV192
2010–05–20

Přehled přednášky

Vlákna a GUI

Vlákna a JNI

Futures & TPE v C++

Ostatní

Vlákna a GUI

- GUI jsou obecně řízená událostmi
  - asynchronní vznik události
  - obvykle se jedno vlákno stará o obsluhu událostí – “event loop”

VLÁKNO GUI

řídící smyčka

CHECK EVENTS

DISPATCH CALLBACKS

(PROCESS REQUESTS)

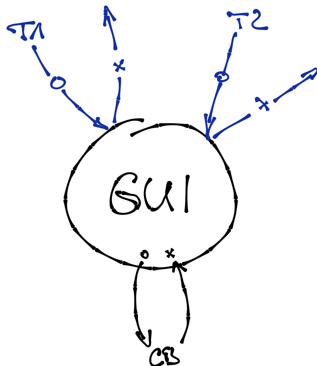
- Problémy událostmi řízeného modelu GUI
  - problém předávání změn z jiných vláken
  - problém s dlouho běžícími úlohami obsluhujícími událost

Vlákna a GUI

- Možnosti synchronizace mezi vlákny a GUI
  1. thread-safe/multi-thread GUI
     - nepoužívá se
     - Graham Hamilton: Multithreaded toolkits: A failed dream?

       `http://weblogs.java.net/blog/kgh/archive/2004/10/multithreaded_t.html`
  2. explicitní zamykání
     - např. Gtk s použitím Gdk zámků
  3. předávání práce GUI vláknu
     - Java SWING, QT, GtkAda Contributions

Explicitní zamykání s GUI



Explicitní zamykání s GUI

- Strategie
  1. před startem GUI inicializuji zamykání
  2. před startem smyčky obsluhy událostí získám zámek
  3. smyčka obsluhy událostí zámek periodicky pouští
  4. jiné vlákno, pokud chce kreslit, musí získat zámek a po dokončení jej pustit
- Problémy
  - ověření, že při všech aktualizacích GUI získávám zámek
  - ověření, že po všech aktualizacích GUI pouštím zámek
  - ověření, že u callbacků *nezískávám a nepouštím* zámky

GtkAda

- Ada binding pro Gtk+, Glib a Gdk
  - prakticky kompletní wrapper z pohledu Gtk+
  - může spolupracovat s tasky
  - 2.14 stabilní, 2.18 v CVS
- Dostupné pro Windows, Linux, MacOS X
- **`http://libre.adacore.com/libre/tools/gtkada/`**

GtkAda

- Inicializace
- Vytvoření smyčky událostí
  - držení zámku v době startu

```
1    Gtk.Main.Set_Locale;
     Gdk.Threads.G_Init;
3    Gdk.Threads.Init;
     Gtk.Main.Init;
5    Init_GUI;
     Gdk.Threads.Enter;
7    Gtk.Main.Main;
     Gdk.Threads.Leave;
9    return;
```

GtkAda

- Modifikace z jiného vlákna
- Zámek pro modifikace

```
   task body Counter is
2  begin
      Main_Loop :
4     loop
         Counter_Monitor.Wait_For_Start;
6        --  because the main thread is waiting for us on termination
         --  we can touch Gtk objects until we signal termination; beware of ogr
8        --  more appropriate is exiting early
         exit Main_Loop when Counter_Monitor.Check_If_Quit;
10       Gdk.Threads.Enter;
         --  Ada95 syntax
12       Gtk.Button.Set_Label (Global_Window.all.Run_Button, -" Stop ");
         Gdk.Threads.Leave;
14       Counter_Loop :
         for I in 0 .. 1000 loop
16          exit Main_Loop when Counter_Monitor.Check_If_Quit;
            if Counter_Monitor.Check_If_Stop then
18             exit Counter_Loop;
            end if;
20          Gdk.Threads.Enter;
            Gtk.Label.Set_Label (Global_Window.all.Counter_Label, -(Integer'Imag
22          Gdk.Threads.Leave;
            delay 1.0;
24       end loop Counter_Loop;
```

GtkAda

- Modifikace z jiného vlákna
- Zámek pro modifikace

```
        Gdk.Threads.Enter;
        --  Ada2005 extended syntax
        Global_Window.all.Run_Button.all.Set_Label (-" Run ");
        Counter_Monitor.Finished;
        Global_Window.all.Run_Button.all.Set_Sensitive (True);
        Gdk.Threads.Leave;
     end loop Main_Loop;
     Counter_Monitor.Finished;
  end Counter;
```

GtkAda

- Callbacky

```
procedure Quit is
begin
   Counter_Monitor.Quit;
   --  BEWARE OF OGRES! May deadlock if lock is not given up!
   Gdk.Threads.Leave;
   Counter_Monitor.Wait_Until_Finished;
   Gdk.Threads.Enter;
   Destroy (Global_Window);
   Gtk.Main.Main_Quit;
end Quit;

procedure On_Quit_Button_Clicked
  (Button : access Gtk.Button.Gtk_Button_Record'Class)
is
begin
   pragma Unreferenced (Button);
   Quit;
end On_Quit_Button_Clicked;
```

GtkAda

- Callbacky

```
procedure On_Run_Button_Clicked
   (Button : access Gtk.Button.Gtk_Button_Record'Class)
is
begin
   if Counter_Monitor.Check_If_Running then
      Button.all.Set_Sensitive (False);
      Counter_Monitor.Stop;
   else
      Counter_Monitor.Start;
   end if;
end On_Run_Button_Clicked;
```

GtkAda

- Callbacky

```
1        Button_Callback.Connect
           (Global_Window.all.Quit_Button,
3           "clicked",
            Button_Callback.To_Marshaller (On_Quit_Button_Clicked'Access),
5           False);
        --  XXX: window delete event should be also handled, but for simplicity
7       --       reasons, it is ommitted.

9        Button_Callback.Connect
           (Global_Window.all.Run_Button,
11          "clicked",
            Button_Callback.To_Marshaller (On_Run_Button_Clicked'Access),
13          False);
```

GtkAda

- Synchronizace mezi GUI vláknem a počítacím vláknem

```
protected Counter_Monitor is
   procedure Start;
   entry Wait_For_Start;
   procedure Stop;
   function Check_If_Stop return Boolean;
   procedure Quit;
   function Check_If_Quit return Boolean;
   procedure Finished;
   function Check_If_Running return Boolean;
   entry Wait_Until_Finished;
private
   Should_Start : Boolean := False;
   Should_Stop : Boolean := False;
   Should_Quit : Boolean := False;
   Is_Running : Boolean := False;
end Counter_Monitor;
```

GtkAda

- Synchronizace mezi GUI vláknem a počítacím vláknem

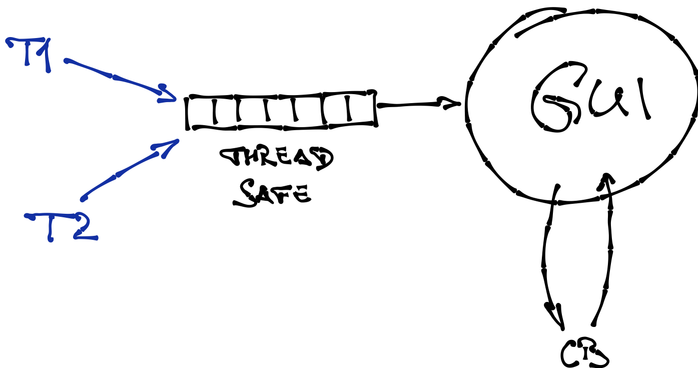
```
1  protected body Counter_Monitor is
      procedure Start is
3     begin
         Should_Start := True;
5     end Start;

7     entry Wait_For_Start
        when Should_Start or Should_Quit is
9     begin
         Should_Start := False;
11       Should_Stop := False;
         --  don't touch Should_Quit, so that it's persistant
13       Is_Running := True;
      end Wait_For_Start;
```

Gtk

`http://www.gnu.org/software/guile-gnome/docs/gdk/html/Threads.html`

Asynchronní předávání do GUI



Asynchronní předávání do GUI

- Strategie
  1. vytvoříme smyčku obsluhy událostí GUI, která volá přímo jednotlivé callbacky
  2. vlákno, které chce aktualizovat GUI předá práci GUI smyčce
  3. do vlákna zpracovávajícího události můžeme zasílat blokujícím (`Request`) nebo neblokujícím (`Send`) způsobem
- Problémy
  - ověření, že při všech aktualizacích GUI používáme předání práce GUI vláknu

GtkAda a Gtk.Main.Router

- GtkAda Contributions: **http://www.dmitry-kazakov.de/ada/gtkada_contributions.htm**
- Inicializace
- Vytvoření smyčky událostí
  - není třeba podpora vláken z Gdk

```
Gtk.Main.Set_Locale;
Gtk.Main.Init;
Gtk.Main.Router.Init;
Init_GUI;
Gtk.Main.Main;
return;
```

GtkAda a Gtk.Main.Router

- Zasílání z vlákna o smyčky událostí

```
task body Counter is
begin
   Main_Loop :
   loop
      Counter_Monitor.Wait_For_Start;
      --  if we don't exit here when appropriate, the application would
      --  deadlock: the GUI task callback is waiting for us while we're
      --  synchronously calling modification of the GUI
      exit Main_Loop when Counter_Monitor.Check_If_Quit;
      Gtk.Main.Router.Request (+Update_GUI_Button_Start'Access);
      Counter_Loop :
      for I in 0 .. 1000 loop
         exit Main_Loop when Counter_Monitor.Check_If_Quit;
         if Counter_Monitor.Check_If_Stop then
            exit Counter_Loop;
         end if;
```

GtkAda a Gtk.Main.Router

- Zasílání z vlákna o smyčky událostí

```
            declare
               Label : aliased Unbounded_String := To_Unbounded_String (Integer'
            begin
               Update_GUI_Label_Callback.Request (Update_GUI_Label'Access, Label
            end;
            delay 1.0;
         end loop Counter_Loop;
         declare
            NR : aliased Null_Record;
         begin
            --  this goes asynchronously
            Update_GUI_Button_End_Handler.Send (Update_GUI_Button_End'Access, NR
         end;
         Counter_Monitor.Finished;
      end loop Main_Loop;
      Counter_Monitor.Finished;
   end Counter;
```

GtkAda a Gtk.Main.Router

- Registrace volání
  - generika
  - komplikované kvůli typům

```
1   procedure Update_GUI_Button_Start is
    begin
3         Gtk.Button.Set_Label (Global_Window.all.Run_Button, -" Stop ");
    end Update_GUI_Button_Start;
5
    type Null_Record is null record;
7   procedure Update_GUI_Button_End (NR : in out Null_Record) is
       pragma Unreferenced (NR);
9   begin
          Global_Window.all.Run_Button.all.Set_Label (-" Run ");
11        Global_Window.all.Run_Button.all.Set_Sensitive (True);
    end Update_GUI_Button_End;
13
    procedure Update_GUI_Label (Label_Access : access Unbounded_String) is
15  begin
       Gtk.Label.Set_Label (Global_Window.all.Counter_Label, -To_String (Label_Ac
17  end Update_GUI_Label;
```

GtkAda a Gtk.Main.Router

- Registrace volání
  - generika
  - komplikované kvůli typům

```
1     --  this is ugly
      type Local_Callback is access procedure;
3     function "+" is
         new Ada.Unchecked_Conversion (Local_Callback, Gtk.Main.Router.Gtk_Callback
5
      -- this is better
7     package Update_GUI_Label_Callback is new Gtk.Main.Router.Generic_Callback_Req

9     package Update_GUI_Button_End_Handler is new Gtk.Main.Router.Generic_Message
```

Java SWING

- Multiplatformní GUI v Javě
- SWING single-thread rule
  - všechny prvky mohou být vytvářeny, měněny a dotazovány pouze z vlákna obsluhujícího události
  - **`SwingUtilities.isEventDispatchThread`** – kontrola, zda jsme ve vlákně obsluhující události
  - **`SwingUtilities.invokeLater`** – předávání **`Runnable`** do vlákna obsluhujícího události
  - **`SwingUtilities.invokeAndWait`** – předávání **`Runnable`** do vlákna obsluhujícího události a zablokuje se do dokončení akce
  - callbacky se řeší pomocí akcí action listener z vlákna obsluhujícího události
  - dlouho běžící callbacky je možno odštípnout do nového vlákna (přímo nebo přes Executory)

Java SWING

- Inicializace

```
1 public class JavaGUI {
2
3     public static void main(String[] args) {
4         CounterDialog dialog = new CounterDialog();
5         dialog.setSize(400,300);
6         dialog.setVisible(true);
7     }
8
9 }
```

Java SWING

- Předávání vláknu událostí SWINGu

```
    @Override
2   public void run() {
        for (int i = 0; i <= 1000; i++) {
4           if (shouldShutdown.get()) {
                break;
6           }
            final String labelText = String.valueOf(i);
8           javax.swing.SwingUtilities.invokeLater(new Runnable() {
                public void run() {
10                  counterLabel.setText(labelText);
                }
12          });
            try {
14              Thread.sleep(1000);
            } catch (InterruptedException ignored) {
16          }
        }
18      runButton.setText("Run");
        runButton.setEnabled(true);
20  }
```

Java SWING

- Callbacky

```
1          buttonRun.addActionListener(new ActionListener() {
               public void actionPerformed(ActionEvent e) {
3                  onRun();
               }
5          });
```

```
       private void onRun() {
2          if (!isRunning.get()) {
               buttonRun.setText("Stop");
4              isRunning.set(true);
               if (counterThread != null) {
6                  try {
                       counterThread.join();
8                  } catch (InterruptedException ignored) {
                   }
10             }
               counterThread = new CounterThread(counterLabel, buttonRun);
12             counterThread.start();
           } else {
14             buttonRun.setEnabled(false);
               counterThread.requestShutdown();
16             counterThread.interrupt();
               isRunning.set(false);
18         }
       }
```

QT

`http://doc.trolltech.com/3.3/threads.html`

OpenGL

- Průmyslový standard specifikující multiplatformní rozhraní pro tvorbu aplikací počítačové grafiky
- Existuje v řadě verzí od 1.0 po poslední 4.0
- Aplikace využívající OpenGL je schopna ±běžet na různém HW
  - OpenGL podporuje různá rozšíření, která spolu se změnami verzí zhoršují poratabilitu
  - Aplikace musí umět detekovat co daná platforma nabízí
- Rasterizaci objektů provádí obvykle HW, existují ale i SW razterizátory (Mesa)

OpenGL pipeline

- OpenGL funkce jsou asynchronní
- OpenGL je stavová
- Návrh scény provádíme:

```
glBegin( GL_POLYGON );           /* Begin issuing a polygon */
glColor3f( 0, 1, 0 );            /* Set the current color to green */
glVertex3f( -1, -1, 0 );         /* Issue a vertex */
glVertex3f( -1, 1, 0 );          /* Issue a vertex */
glVertex3f( 1, 1, 0 );           /* Issue a vertex */
glVertex3f( 1, -1, 0 );          /* Issue a vertex */
glEnd();                         /* Finish issuing the polygon */
```

- Problematické při použití vláken

OpenGL kontext

- Kontex OpenGL není zachycen OpenGL specifikací, je tedy implementačně závislý
- Kontex uchovává stav a další informace pro rasterizér
- Existují rozšíření OpenGL pro manipulaci s kontexty
  - WGL – `wglCreateContext`, `wglMakeCurrent`, `wglDeleteContext`
  - GLX – `glXCreateNewContext`, `glXMakeCurrent`, `glXDestroyContext`
  - Pozor na použití glXDestroyContext – OpenGL je asynchronní!
- Pouze jeden kontext může být aktivní v rámci 1 vlákna
- Kontext je často uložen v TLS

OpenGL kontext a vlákna

- S implicitním kontextem
  - OpenGL na MacOS dovoluje přístup k GL funkcím z více vláken
  - Windows ohlásí resource busy
  - Linux (s Nvidia drivery) končí segmentation fault
  - Obecně je u vláken s implicitním kontextem problém, že kontext má právě master vlákno
  - Nelze triviálně předat kontext jinému vláknu (zejména u GLX)
- Různá vlákna si mohou vytvořit svůj kontext
  - Nemohou pak ale kreslit jednu společnou scénu přímo (lze přes nepřímý rendering do bufferu)
  - U WGL to často znamená, že každé vlákno kreslí do jiného okna

OpenGL kontext a vlákna

- Změna kontextu na jiné než master vlákno
  - Při použití `libSDL` triviálně tak, že zavoláme `SDL_init()` z ne-master vlákne, o zbytek se postará `libSDL`
  - Implicitní kontext vytváří GLX rozšíření Xserveru samo o sobě
  - Jediná cesta pro GLX je reload GLX z vlákna, které má kreslit
  - Návod lze najít např. ve zdrojových kódech `libSDL` `src/video/x11/SDL_x11gl.c`

Java Native Interfaces

- Volání nativních metod z Javy
- Struktura JNI volání

```
/* C */
JNIEXPORT void JNICALL Java_ClassName_MethodName
  (JNIEnv *env, jobject obj, jstring javaString)
{
    //ziskani nativniho retezce z javaString
    const char *nativeString = (*env)->GetStringUTFChars(env, javaString, 0);
    ...
    //nezapomenout uvolnit!
    (*env)->ReleaseStringUTFChars(env, javaString, nativeString);
}

// C++
JNIEXPORT void JNICALL Java_ClassName_MethodName
  (JNIEnv *env, jobject obj, jstring javaString)
{
    //ziskani nativniho retezce z javaString
    const char *nativeString = env->GetStringUTFChars(javaString, 0);
    ...
    //nezapomenout uvolnit!
    env->ReleaseStringUTFChars(javaString, nativeString);
}
```

Vlákna a JNI

- Ukazatel na **`JNIEnv`** je platný pouze z vlákna, jemuž je přiřazen
  - nelze předávat mezi vlákny
  - stejný při opakovaných voláních v témže vlákně
- Lokální reference nesmí opustit vlákno
  - lokální reference jsou platné pouze v rámci daného volání
    - nelze se je uschovávat ve **`static`** proměnných
  - převést na globální, pokud je třeba (**`NewGlobalRef`**)
  - globální reference vylučují objekt z garbage collection
  - existují slabé lokální reference (**`NewWeakGlobalRef`**)
    - umožňují garbage collection odkazovaného objektu
    - potřeba kvůli class unloading
    - musí se kontrolovat, zda odkazovaný objekt existuje (**`IsSameObject`** s NULL parametrem)

Vlákna a JNI

```
static jclass stringClass = NULL;
...
if (stringClass == NULL) {
    jclass localRefCls =
        (*env)->FindClass(env, "java/lang/String");
    if (localRefCls == NULL) {
        return NULL; /* exception thrown */
    }
    /* Create a global reference */
    stringClass = (*env)->NewGlobalRef(env, localRefCls);

    /* The local reference is no longer useful */
    (*env)->DeleteLocalRef(env, localRefCls);

    /* Is the global reference created successfully? */
    if (stringClass == NULL) {
        return NULL; /* out of memory exception thrown */
    }
}
...
// potreba explicitne mazat
if (terminate) {
   (*env)->DeleteGlobalRef(env, stringClass);
}
```

Zdroj: JNI Book

Vlákna a JNI

- Použití monitorů
  - vždy monitor uvolnit

```
synchronized (obj) {
    ...                    // synchronized block
}
```

```
if ((*env)->MonitorEnter(env, obj) != JNI_OK) ...;
...
if ((*env)->ExceptionOccurred(env)) {
    ... /* exception handling */
    /* remember to call MonitorExit here */
    if ((*env)->MonitorExit(env, obj) != JNI_OK) ...;
}
... /* Normal execution path. */
if ((*env)->MonitorExit(env, obj) != JNI_OK) ...;
```

Zdroj: JNI Book

Vlákna a JNI

- Wait/Notify
  - generické volání metod (**GetMethodID**, **CallVoidMethod**)
  - potřeba držet monitor

```
/* precomputed method IDs */
static jmethodID MID_Object_wait;
static jmethodID MID_Object_notify;
static jmethodID MID_Object_notifyAll;

void
JNU_MonitorWait(JNIEnv *env, jobject object, jlong timeout) {
    (*env)->CallVoidMethod(env, object, MID_Object_wait, timeout);
}

void
JNU_MonitorNotify(JNIEnv *env, jobject object) {
    (*env)->CallVoidMethod(env, object, MID_Object_notify);
}

void
JNU_MonitorNotifyAll(JNIEnv *env, jobject object) {
    (*env)->CallVoidMethod(env, object, MID_Object_notifyAll);
}
```

Zdroj: JNI Book

Vlákna a JNI

- Explicitní získání **`JNIenv`** pro stávající vlákno
  - např. callback volaný OS
  - odkaz na JavaVM lze předávat mezi voláními a vlákny
  - získání odkazu např. **`JNI_GetCreatedJavaVMs`** nebo **`GetJavaVM`**

```
1  JavaVM *jvm; /* already set */
2
3  f() {
4      JNIEnv *env;
5      (*jvm)->AttachCurrentThread(jvm, (void **)&env, NULL);
6      ... /* use env */
7  }
```

Zdroj: JNI Book

Vlákna a JNI

- Mapování vláknového modelu OS a JVM
  - záleží, zda pro danou platformu JVM podporuje nativní vlákna
  - závisí od dané platformy i od daného JVM
- Můžeme implementovat v Javě afinitu k CPU
  - závisí od dané platformy i od daného JVM
  - `sched_setaffinity(gettid(), ...)`
  - v praxi může pro danou platformu dobře fungovat

Futures & TPE v C++

- Futures nabízí knihovna **libboost**
  http://www.boost.org
  - Nabízí ± stejná primitiva jako **pthread**
  - Navíc nabízí některé pokročilejší nástroje pro vlákna

```
#include <boost/thread.hpp>
#include <iostream>

using namespace std;

void hello()
{
        cout << "Pocitam." << endl;
}

int main(int argc, char* argv[])
{
        boost::thread thrd(&hello);
        thrd.join();
        return 0;
}
```

Futures & TPE v C++

- Futures

```
#include <boost/thread.hpp>
#include <iostream>
#include <string>

using namespace std;

string hello()
{
        cout << "FT: Pocitam." << endl;
        sleep(5);
        cout << "FT: Vypocet hotov." << endl;
        return "12345";
}

int main(int argc, char* argv[])
{
        boost::packaged_task<string> pt(hello);
        boost::unique_future<string> fi=pt.get_future();

        boost::thread task(boost::move(pt));
        cout << "Master: vytvoreno vlakno." << endl;
        cout << "Master: Cekam na vysledek." << endl;
        fi.wait();
        cout << "Mam vysledek: " << fi.get() << endl;
        return 0;
}
```

Futures & TPE v C++

- Samotná knihovna `libboost` nemá threadpools
- `http://threadpool.sourceforge.net/` implementuje threadpools v C++ nad `libboost`

Futures & TPE v C++

> Anybody who tells me that STL and especially Boost are stable and portable is just so full of bullshit that it's not even funny.
>
> – *Linus Torvalds*

Ostatní

- Odevzdávání „projektu“
  - Odevzdavarny/Projekt
  - UCO-Jmeno-Prijmeni.{zip,tgz,...}
- Organizace zkoušky
  - diskuse řešení projektu
  - minimálně 2 obecné otázky
  - minimálně 1 specializační (Java/C/Ada)