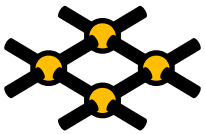
Vláknové programování
část V

Lukáš Hejmánek, Petr Holub

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



Laboratoř pokročilých síťových technologií

PV192

2014–03–25

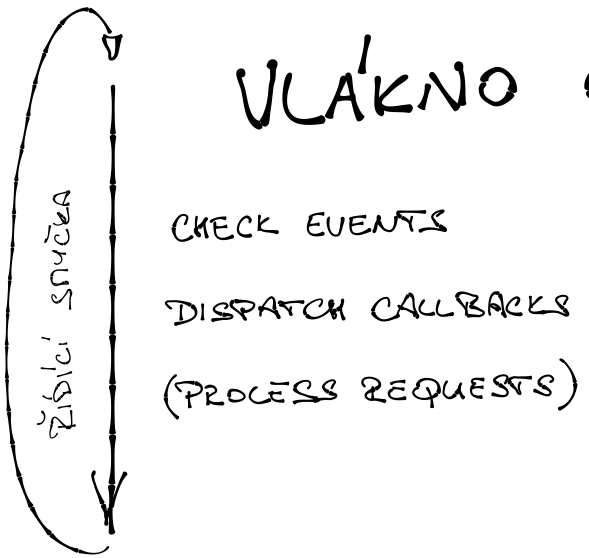
Přehled přednášky

Vlákna a GUI

OpenGL

Vlákna a GUI

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



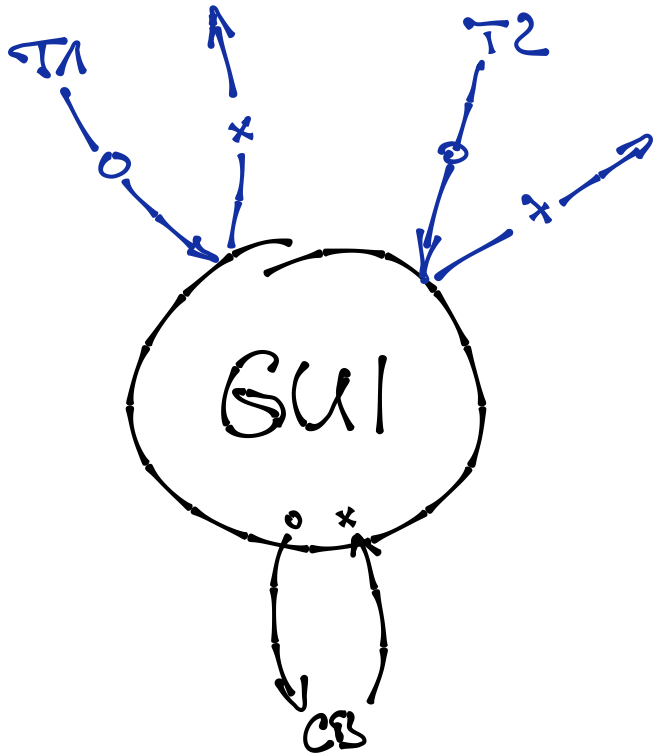
- Problémy událostmi řízeného modelu GUI
  - problém předávání změn z jiných vláken
  - problém s dlouho běžícími úlohami obsluhujícími událost

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



o LOCK

x UNLOCK

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
begin
   Main_Loop :
   loop
      Counter_Monitor.Wait_For_Start;
      --  because the main thread is waiting for us on termination
      --  we can touch Gtk objects until we signal termination; beware of ogr
      --  more appropriate is exiting early
      exit Main_Loop when Counter_Monitor.Check_If_Quit;
      Gdk.Threads.Enter;
      --  Ada95 syntax
      Gtk.Button.Set_Label (Global_Window.all.Run_Button, -" Stop ");
      Gdk.Threads.Leave;
      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;
         Gdk.Threads.Enter;
         Gtk.Label.Set_Label (Global_Window.all.Counter_Label, -(Integer'Imag
         Gdk.Threads.Leave;
         delay 1.0;
      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
2  begin
      Counter_Monitor.Quit;
4     --  BEWARE OF OGRES! May deadlock if lock is not given up!
      Gdk.Threads.Leave;
6     Counter_Monitor.Wait_Until_Finished;
      Gdk.Threads.Enter;
8     Destroy (Global_Window);
      Gtk.Main.Main_Quit;
10 end Quit;

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

GtkAda

- Callbacky

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

GtkAda

- Callbacky

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

      Button_Callback.Connect
        (Global_Window.all.Run_Button,
         "clicked",
         Button_Callback.To_Marshaller (On_Run_Button_Clicked'Access),
         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

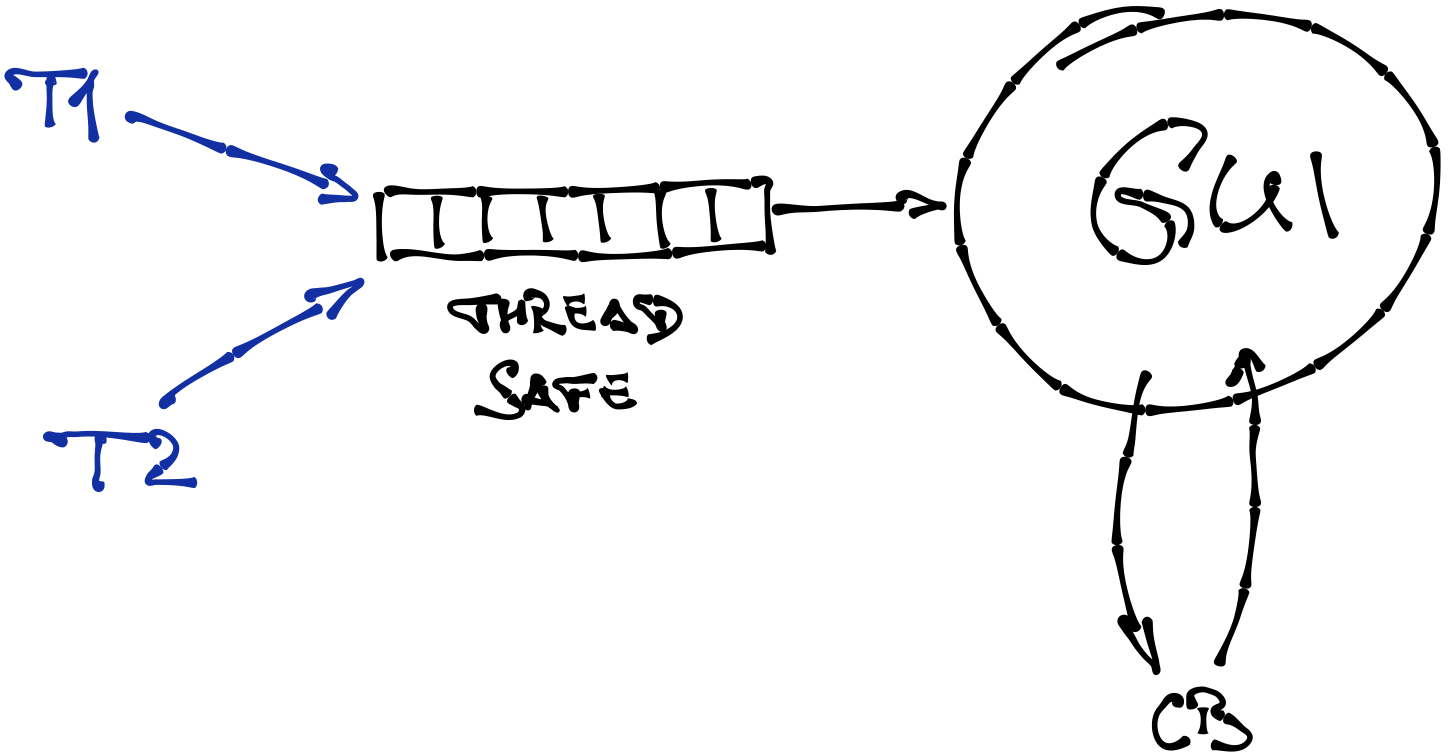
```
protected body Counter_Monitor is
   procedure Start is
   begin
      Should_Start := True;
   end Start;

   entry Wait_For_Start
     when Should_Start or Should_Quit is
   begin
      Should_Start := False;
      Should_Stop := False;
      --  don't touch Should_Quit, so that it's persistant
      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

```
1   task body Counter is
    begin
3      Main_Loop :
       loop
5         Counter_Monitor.Wait_For_Start;
          --  if we don't exit here when appropriate, the application would
7         --  deadlock: the GUI task callback is waiting for us while we're
          --  synchronously calling modification of the GUI
9         exit Main_Loop when Counter_Monitor.Check_If_Quit;
          Gtk.Main.Router.Request (+Update_GUI_Button_Start'Access);
11        Counter_Loop :
          for I in 0 .. 1000 loop
13           exit Main_Loop when Counter_Monitor.Check_If_Quit;
             if Counter_Monitor.Check_If_Stop then
15              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 {

3     public static void main(String[] args) {
          CounterDialog dialog = new CounterDialog();
5         dialog.setSize(400,300);
          dialog.setVisible(true);
7     }

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/4.7/threads.html`

- rozlišování reentrant vs. thread-safe
- **`QThread`** reprezentuje vlákno
- **`QObject`** je reentrantní, stejně jako řada odvozených tříd
- avšak odvozené GUI třídy **`QWidget`** reentrantní nejsou!
  - použití pouze z hlavního vlákna
- per-thread event loop
  - pro třídy jako **`QTimer`**, **`QTcpSocket`** – ne pro GUI

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

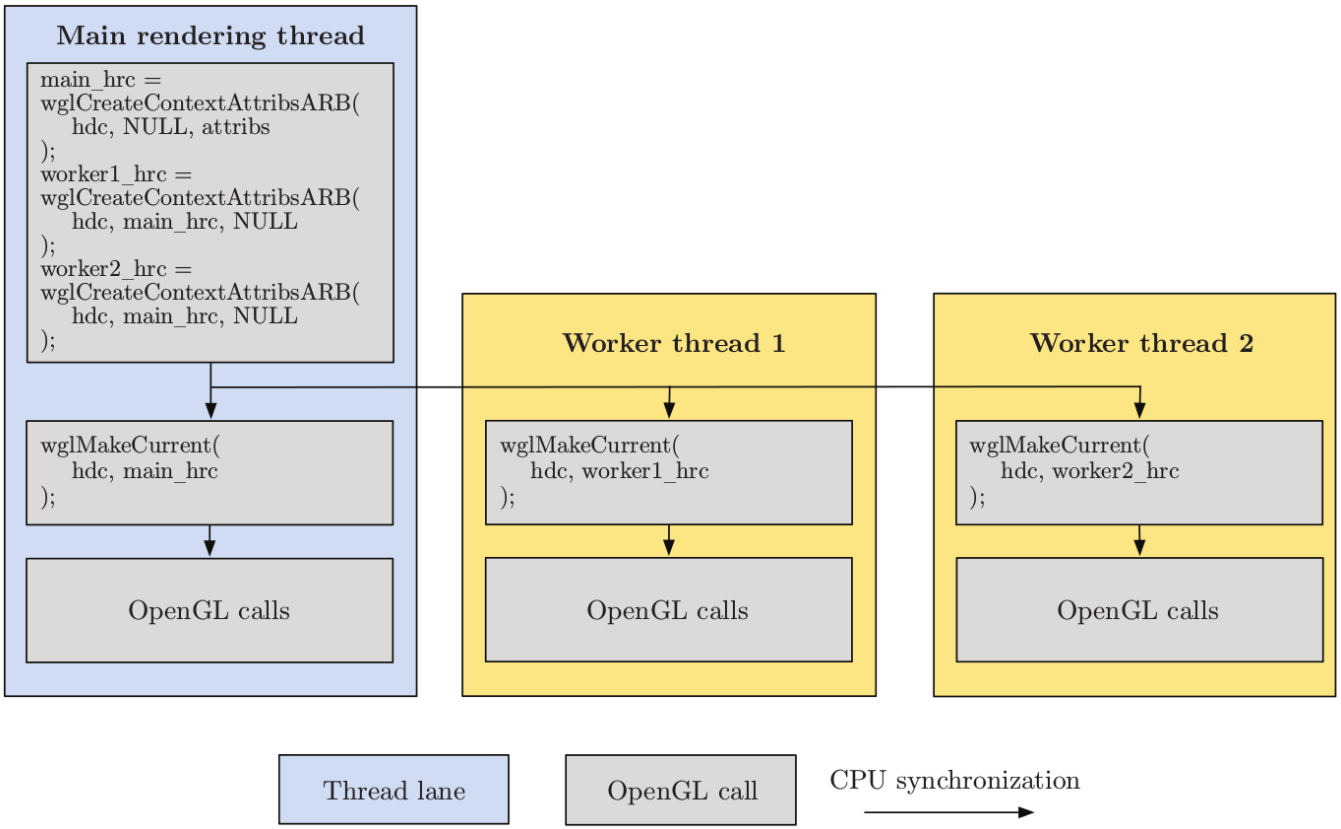
OpenGL kontext

- 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 přistup 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
- Práce s OpenGL z více vláken z výkonnostních důvodů
  - využití Pixel Buffer Objects (PBO) a asynchronních přenosů
  - **`http://www.seas.upenn.edu/~pcozzi/OpenGLInsights/OpenGLInsights-AsynchronousBufferTransfers.pdf`**

OpenGL kontext pro Windows



OpenGL kontext pro Linux

```
void *
foo(void *ctx)
{
        GLXContext mainCtx = (GLXContext)ctx;

        GLXContext myCtx = glXCreateContext(dpy, vi, mainCtx, GL_TRUE)

        glXMakeCurrent (dpy, win, ctx);

        glClearColor (sl/6.0, sl/6.0, 1, 1);
        glClear (GL_COLOR_BUFFER_BIT);
        glXSwapBuffers (dpy, win);
        glFlush();
        XFlush(dpy);
}
```

OpenGL kontext pro Linux

```
int
main()
{
        [...]
        GLXContext ctx = glXCreateContext(dpy, vi, 0, GL_TRUE);

        glXMakeCurrent (dpy, win, ctx);

        pthread_create(&t1, NULL, foo, ctx);
        pthread_create(&t2, NULL, foo, ctx);
        pthread_create(&t3, NULL, foo, ctx);

        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        pthread_join(t3, NULL);

        ctx = glXGetCurrentContext();
        glXDestroyContext(dpy, ctx);
}
```

OpenGL kontext a vlákna

- Alternativní 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`