

Programování pro X Window System

Martin Beran

17. února 2004

Slide 1

SISAL MFF UK, Malostranské nám. 25, 118 00 Praha 1

`beran@ms.mff.cuni.cz`

`http://www.ms.mff.cuni.cz/~beran/`

- Předmět je 2/2 Zk/Z.
- Zápočty budou za referát nebo za zápočtový program.
- Průběh zkoušky:
 1. předvedení zápočtového programu, diskuse na téma zápočtového programu nebo referátu
 2. 2teoretické otázky
- Sledujte stránku `http://www.ms.mff.cuni.cz/~beran/vyuka/X/`
- Předpoklady:
 - uživatelská znalost OS Unix nebo podobných (Linux)
 - programování v C pro Unix
 - základní uživatelská znalost X

Slide 2

1. Úvod

X Window System je...

- standardní unixové grafické uživatelské rozhraní (GUI, graphical user interface)
- oddělený od operačního systému
- distribuovaný
- otevřený:
 - publikována úplná specifikace X protokolu a API (application programming interface)
 - rozšiřitelný protokol
 - vrstevnatá modulární architektura
 - volně dostupná referenční implementace
 - standard udržovaný sdružením výrobců X.Org

Slide 3

- distribuovanost:
 - aplikace běžící na jiném počítači než kde sedí uživatel
 - na jedné obrazovce okna programů z různých počítačů
 - uživatelé z různých míst pracující s programy na jednom počítači
- optimalizace (sdílená paměť), když je klient a server na jednom počítači
- rozšiřitelnost – protocol/server extensions
- vrstvy: X protocol, Xlib, toolkity, aplikace, window managery
- XFree86, komerční verze
- plný název „X Window System“ zkracován na „X“ raději než „X Windows“

Slide 4

Obsah přednášky

1. Úvod – literatura, historie, architektura X, styl programování GUI
2. toolkity GTK+, Qt
 - architektura toolkitů
 - přehled widgetů
 - podpůrné knihovny
 - komunikace mezi procesy (selections, drag&drop)
 - psaní nových widgetů
3. Xlib
 - základní pojmy a principy X a Xlib, X protokol
 - programování s použitím Xlib
 - kreslení
 - zpracování událostí
 - konfigurační soubory, window management, rozšíření X

- U GTK+ se zaměříme se na programování v C, i když existují API i pro jiné jazyky, např. `gtkmm` pro C++.
- Budeme se bavit o GTK+ verzích 2.0 a 2.2 a o Qt 3.1.
- Na přednášce budeme popisovat vlastnosti a funkce X, GTK+ a Qt, na cvičení (a částečně i na přednášce) to doplníme praktickými ukázkami konkrétních programů.
- Předpokládají se znalosti: editace zdrojového textu, volání kompilátoru a linkeru, `make`, ladění, základní znalost prostředí X na uživatelské úrovni.

Literatura

1. The definitive Guides to the X Window System.
O' Reilly & Associates
2. Havoc Pennington: GTK+/GNOME Application Development
New Riders Publishing, ISBN 0-7357-0078-8
<http://developer.gnome.org/doc/GGAD/>
3. GTK+: <http://www.gtk.org/>
4. GNOME: <http://www.gnome.org/>
5. Qt: <http://www.trolltech.com/>
6. KDE: <http://www.kde.org/>

- Stránky GTK+ a GNOME obsahují velké množství dokumentace:
 - referenční manuály GTK+ a dalších pomocných knihoven (GLib, GObject, Pango, ATK, GdkPixbuf, GDK, GTK)
 - Tony Gale, Ian Main & the GTK team: GTK+ 2.0 Tutorial
 - dokumentace pro vývojáře GNOME
- Dokumentace GTK+ je často neúplná, je třeba kombinovat několik zdrojů informací, experimentovat a studovat zdrojové kódy. Od verze 1.2 se dokumentace značně zlepšila.
- Stránky firmy Trolltech obsahují kompletní, velmi kvalitní dokumentaci toolkitu Qt. Pro běžné využití toolkitu není třeba hledat další dokumenty ani se dívat do zdrojových kódů.
- Stránky projektu KDE obsahují vývojářskou dokumentaci KDE.

Slide 6

Verze a licence

- **X11**
 - X11R6.6
 - implementace zveřejněná konsorciem X.Org je volně použitelná pro libovolný účel
 - XFree86 má podobnou volnou licenci
 - komerční implementace X používají různé licence
- **GTK+**
 - stabilní verze 2.2, předchozí 1.2, vývojová verze 2.3
 - licence GNU LGPL
- **Qt**
 - stabilní verze 3.2, vývojová 3.3.0 Beta 1
 - variantně komerční licence a GPL

- Informace o verzích jsou platné k datu tohoto slidu 30. ledna 2004.
- Licence X je velmi podobná BSD licenci, pod kterou je dostupné např. FreeBSD.
- Budeme se zabývat Qt 3.1, novinky z 3.2 a 3.3 vynecháme.

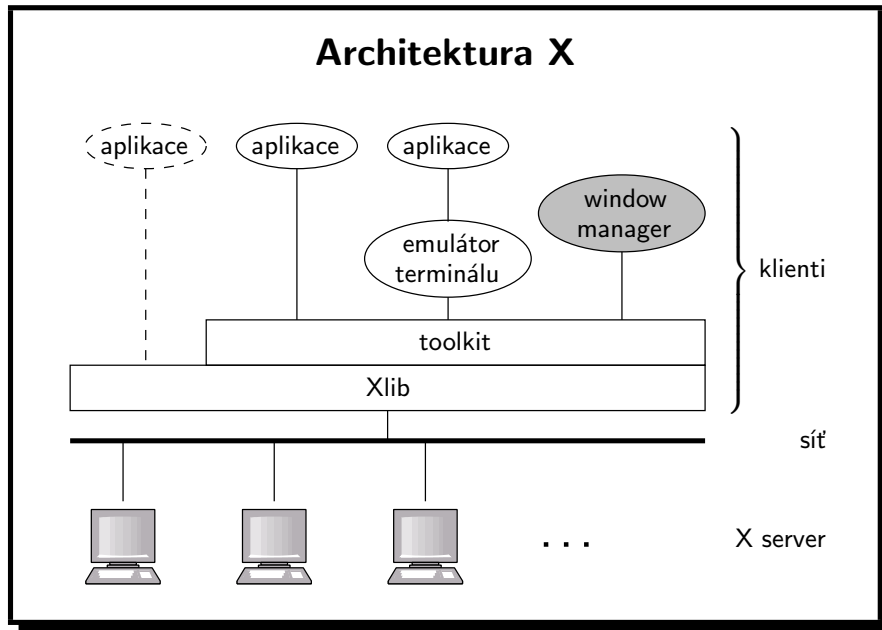
Historie X

| | |
|------|---|
| 1983 | projekt Athena na MIT v Laboratory for Computer Science |
| 1984 | zrod X – verze X1 až X6 |
| 1985 | zveřejnění verze X9 |
| 1986 | verze X10 Release 4 základem pro komeční produkty |
| 1987 | verze X11R1 – současná verze X protokolu |
| 1988 | X11R2, založeno MIT X Consortium |
| 1993 | vzniklo X Consortium, nástupce předchozího MIT konsorcia |
| 1994 | X11R6, start projektu XFree86 |
| 1999 | X Consortium se stalo pod názvem X.Org částí The Open Group |
| 2001 | verze X11R6.6 |

Slide 7

- Athena – distribuovaný systém pro potřeby vědy a výuky na MIT, X je jeho součástí (další je např. Kerberos)
- Datování je nejisté, nepodařilo se mi najít přesnou chronologii vývoje X a souvisejících organizací.
- The Open Group – konsorcium pro otevřené systémy, spravuje např. standardy pro systém Unix

Slide 8



- X server je to, u čeho sedí uživatel.
- Klient je jeho aplikace, která může běžet na vzdáleném a mnohem silnějším počítači.
- Server a klient nejsou obráceně, protože klient obecně není „to, co běží na straně uživatele“ a server „to, k čemu uživatel přistupuje prostřednictvím klienta“. Je nutné si uvědomit, že klient je proces, který vyžaduje nějaké služby (zde zobrazování svého výstupu a zasílání zpráv o reakcích uživatele), a server je proces poskytující tyto služby.

Slide 9

Komponenty X

- **X server** ... proces, který zajišťuje výstup na obrazovku a vstup z klávesnice, myši, popř. dalších vstupních zařízení (např. tablet)
 - Na jednom počítači může být více X serverů (**display**), každý může mít několik obrazovek (**screen**).
- **X klient** ... aplikace, která používá X pro své GUI
- **X protokol** ... síťový protokol pro komunikaci mezi X serverem a X klientem
- **Xlib** ... API knihovna, překládá volání funkcí X na zprávy X protokolu
 - funkce na úrovni základní manipulace s okny, kreslení a přijímání událostí od klávesnice a myši
 - server informuje aplikaci pomocí událostí o akcích uživatele

- Rozšíření Xinerama umožňuje spojit více monitorů do jedné logické obrazovky.
- Server typicky běží na pracovní stanici (PC) s grafickým displejem nebo na speciálním X terminálu.
- Pokud je klient i server na jednom počítači, jsou to stále samostatné procesy, ale pro urychlení mohou komunikovat přes lokální (AF_UNIX) socket a sdílenou paměť místo po síti.
- Xlib neposkytuje GUI, ale jen funkce pro jeho vytvoření. Lze tedy říci „nakresli obdélník, do něj text a čekej, až uživatel klikne myší“, ale ne „vytvoř tlačítko, které při stisku vygeneruje událost XYZ“.

Slide 10

Komponenty X (2)

- **toolkity (X Toolkit, Motif, GTK+, Qt)** ... poskytují prvky uživatelského rozhraní (tlačítka, menu, scrollbar, atd.)
- **window manager** ... kreslí rámečky oken, řídí manipulace s okny (posouvání, zvětšování, zavírání...)
 - samostatná aplikace, uživatel si může zvolit window manager, který mu nejvíce vyhovuje (twm, mwm, olwm, fwm95, Enlightenment, kwin...)
- **terminálový emulátor (xterm)** ... pomocí pseudoterminálů emuluje terminál pro aplikace s textovým vstupem a výstupem (především shell)
- Existují i celá integrovaná grafická prostředí, jako CDE, KDE nebo GNOME.

- Tím, že základní funkce pro interakci s uživatelem, vlastní GUI a window manager jsou navzájem oddělené, se dosahuje velké flexibility.
- Okenní systém je zcela oddělen od základního operačního systému

Slide 11

Spuštění aplikace X

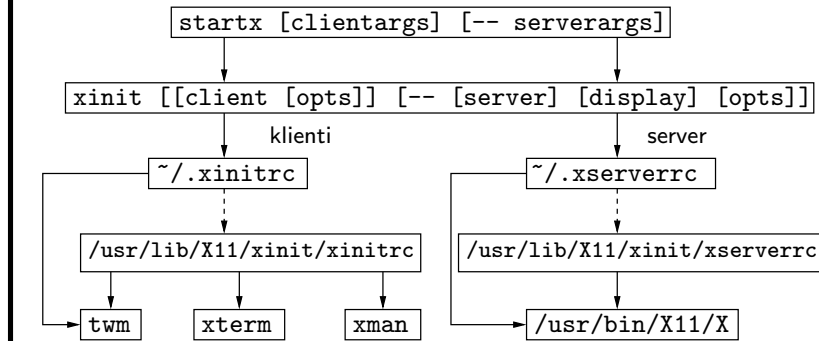
- Buď pomocí menu v prostředí X, nebo ze shellu
- Proměnná prostředí DISPLAY určuje, s kterým X serverem bude aplikace pracovat
 - DISPLAY=:0.0 ... lokální X server na stejném počítači, kde běží aplikace
 - DISPLAY=omega.ms.mff.cuni.cz:1.0 ... display číslo 1, obrazovka č. 0 na počítači omega.ms.mff.cuni.cz
 - obecně DISPLAY=[host]:display[.screen]
- Aplikace, která se připojí k X serveru má přístup ke všemu, co se na X serveru děje, může manipulovat i s okny ostatních aplikací. Proto je potřeba omezit přístup k serveru:
 - xhost ... povolení přístupu pouze z některých počítačů
 - xauth ... povolení pouze pro aplikace, které znají klíč (MIT-MAGIC-COOKIE)

- Jedna aplikace může současně spolupracovat s několika X servery.
- Pokud na jednom počítači spouští klientské procesy více uživatelů, je bezpečnější používat MIT-MAGIC-COOKIE.

Start X

- Z textové systémové konzole (ze shellu) se X spustí obvykle příkazem `startx`.

Slide 12

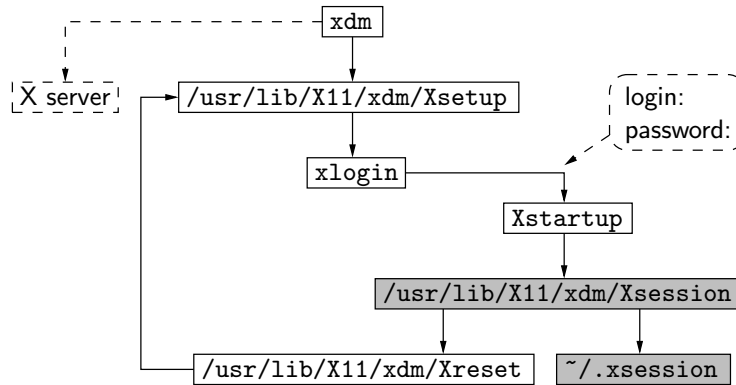


- `startx` je skript, lze ho přizpůsobit podle potřeby. Zobrazené schéma odpovídá standardnímu chování `startx`, kdy se nejdříve hledají startovací skripty v uživatelském a pak v systémovém adresáři.
- Program `xinit` provede vlastní spuštění serveru a klientů. Když skončí proces `client` (obvykle skript, který spustí jednotlivé klienty na pozadí a nakonec provede `exec` na window manager), skončí server i `xinit`.
- X server lze spustit i samostatně a pak spouštět klienty.

Slide 13

X Display Manager (xdm)

- Démon xdm dovoluje logování na grafických stanicích i na X terminálech přímo do X, bez nutnosti loginu na textovém terminálu a následného spuštění xinit.



- Skript Xsession (a z něho spouštěné programy) se spouští s právy uživatele, který se loguje, vše ostatní běží s právy roota.
- Po ukončení Xreset je X server buď resetován, nebo ukončen a znovu nastartován.
- Grafická prostředí GNOME a KDE mají i své varianty display manageru gdm, resp. kdm.

Událostmi řízené programování

Klasické řízení

- vychází z vnitřní logiky programu
- program běží a občas, když potřebuje, provede I/O operaci

Řízení událostmi

- vychází z interakce s okolím (s uživatelem)
- kolekce handlerů událostí a hlavní smyčka pro zpracování událostí
- program čeká na příchod události, zpracuje ji a čeká na další

```
while(event = wait_for_event())  
    process_event(event);
```

- Program používající klasické řízení výpočtu občas něco vypíše nebo vyzve uživatele, aby něco zadal. Tento přístup nutí uživatele do určitého pevně daného postupu interakce s programem.
- Program řízený událostmi v každém okamžiku nabízí uživateli mnoho možností, co udělat (např. v jakém pořadí vyplnit položky dialogu), a pružně reaguje na akce uživatele.
- Platí, že handlers událostí mají běžet jen krátkou dobu a co nejrychleji se vrátit do smyčky zpracování událostí, aby z pohledu uživatele nenastávala situace, kdy aplikace po nějakou dobu nereaguje. Handler by neměl volat funkce, které mohou zablokovat proces (např. blokuující I/O operace). Pro tyto účely je obvykle k dispozici obdoba systémového volání `select()`: proces si zaregistruje file deskriptor, pro který chce dostávat události informující o připravenosti deskriptoru pro čtení nebo zápis. V handleru události se zavolá neblokuující operace čtení nebo zápisu a pokud nebyla přenesena všechna data, nechá se deskriptor zaregistrovaný a v některé další iteraci cyklu událostí I/O operace pokračuje. Pokud je potřeba provést časově náročný výpočet, je třeba ho buď rozdělit na menší části a při jednom zavolání handleru vykonat jen jednu část, nebo celý výpočet přesunout do samostatného procesu nebo vlákna.
- Řízení událostmi je obvyklé pro grafická uživatelská rozhraní, v X ho používá Xlib, GTK+, Qt i ostatní toolkity.

Slide 15

Objektově orientované programování

- Přirozené pro GUI – objekty odpovídají jednotlivým prvkům rozhraní (okna, menu, tlačítka, apod.)
- Různé typy prvků GUI mají společné vlastnosti a tím implikují hierarchii dědičnosti příslušných objektů (např. radio button je odvozené z check boxu a ten je odvozen z obyčejného tlačítka).
- Použití objektů pro uživatelské rozhraní nevyžaduje objektově orientovaný programovací jazyk, např. GTK+ je napsané v C.
- Objektová orientace je běžná pro grafická uživatelská rozhraní, ale není nezbytná.
- Toolkity v X jsou obvykle objektově orientované, ale Xlib není.

- Qt je napsané v C++, proto je kód aplikace typicky mnohem stručnější a přehlednější než v GTK+.
- Použití objektového přístupu přispívá k lepší struktuře programu používajícího grafické rozhraní.
- V X je oddělená logická struktura rozhraní daná objekty (widgety) uvnitř programu a jeho fyzická reprezentace představovaná zdroji (resources) X serveru (okna, fonty, grafické kontexty, atd.).

Slide 16

2. GTK+

Slide 17

GTK+

- **The GIMP Toolkit** – původně vyvinut pro GIMP
- Budeme se zabývat současnou stabilní verzí GTK+ 2.0, resp. 2.2.
- Objektově orientovaný toolkit
- Napsaný v C, existují i varianty API (language bindings) pro další jazyky: Ada, C++, Perl, Python, aj.
- **Glade** – nástroj pro grafický návrh částí uživatelského rozhraní, generuje kostru aplikace

- Verze 2.2 je binárně i na úrovni API kompatibilní s 2.0.
- V C++ lze buď použít API pro C nebo *gtkmm* (dříve *GTK--*), které poskytuje rozhraní ve formě C++-ových tříd a metod.
- Budeme popisovat pouze principy a základní funkce, detaily a další funkce lze najít v referenčních manuálech, ukázkových programech nebo přímo ve zdrojových textech GTK+.

Slide 18

Základní pojmy

- **widget** ... prvek uživatelského rozhraní (např. check box nebo textový řádek), v programu je reprezentován objektem třídy `GtkWidget` nebo z ní odvozené třídy
- **objekt** ... instance třídy `GObject` (kořen hierarchie tříd) nebo nějaké odvozené třídy (widget je speciální případ objektu)
- **okno (GTK+)** ... widget `GtkWindow` pro toplevel okno (hlavní okno programu, dialog, popup)
- **okno (X/GDK)** ... zdroj X serveru, zobrazující widget na obrazovce a přijímající události; každý widget může mít jedno nebo více X oken nebo používá X okno rodiče
- **signál, událost** ... informace, na kterou může program reagovat

- X okna existují na X serveru. Klient s nimi pracuje pomocí `Xlib` a odkazuje se na ně pomocí ID.
- Každý widget potřebuje X okno, aby mohl někam kreslit. Jednoduché widgety, které nepotřebují přijímat události (např. šipka nebo statický text) nemají vlastní X okno a kreslí do okna rodiče. Pokud pro widget bez okna chceme přijímat události, vložíme ho do widgetu `EventBox`.

Signály, události

- Widget emituje **signál**, jestliže se s ním stane něco důležitého, na co program může chtít zareagovat (např. uživatel vybral položky menu nebo zaškrtnl check box).
- X server posílá programu **události** o aktivitě uživatele, X serveru nebo window manageru, na kterou by měl program reagovat.
- Události jsou hlavní smyčkou `gtk_main()` přeloženy na signály a dál zpracovány pomocí **handlerů signálů**.
- Handlerly událostí vrací boolovskou hodnotu.
 - FALSE = událost se bude dál zpracovávat a případně propagovat do rodičovského widgetu
 - TRUE = událost byla zpracována, nepropaguje se ani se nespouští další handlerly

- Události přicházejí do programu z vnějšku, signály generuje program sám. Zdroje událostí a příklady jimi vyvolaných událostí:
 - *uživatel* – stisk klávesy, pohyb myši
 - *X server* – žádost o překreslení oblasti okna
 - *window manager* – zavření, posun nebo změna velikosti okna
- Události se zpracovávají asynchronně – událost je přijata ze zdroje vně programu a uložena do fronty. Po nějaké době ji `gtk_main()` vyzvedne z fronty a zavolá handler.
- Signály se zpracovávají synchronně – při zavolání funkce emitující signál se provedou všechny handlerly registrované pro tento signál a teprve pak emitující funkce skončí.
- Událost nejprve způsobí vyslání signálu "**event**". Když není událost ošetřena v reakci na tento signál, vyšle se specifický signál podle typu události, např. při stisku tlačítka myši je to "**button_press_event**". Pokud událost pochází z klávesnice nebo myši a opět není ošetřena, propaguje se do rodičovského widgetu. Jestliže ani rodič událost neošetří, propaguje se postupně dál až k toplevel oknu. Nakonec se ještě pošle signál "**event-after**".
- Jestliže některý handler událost ošetří (vrátí TRUE), nevolají se pro tuto událost žádné další handlerly.
- Handler se registruje pomocí `g_signal_connect()` vždy pro konkrétní objekt (obvykle widget) a volá se pouze pro signály generované tímto objektem.
- Widget může mít ke stejnému signálu připojeno několik handlerů, volají se postupně.
- Signál lze vygenerovat funkcí `g_signal_emit_by_name()`.

Hello World

```
int main(int argc, char *argv[])
{
    GtkWidget *window, *button;
    gtk_init(&argc, &argv);
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    g_signal_connect(G_OBJECT(window), "delete_event",
                     G_CALLBACK(delete_event), NULL);
    g_signal_connect(G_OBJECT(window), "destroy",
                     G_CALLBACK(destroy), NULL);
    button = gtk_button_new_with_label("Hello World");
    g_signal_connect_swapped(G_OBJECT(button), "clicked",
                              G_CALLBACK(gtk_widget_destroy),
                              G_OBJECT(window));
    gtk_container_add(GTK_CONTAINER(window), button);
    gtk_widget_show(button); gtk_widget_show(window);
    gtk_main();
}
```

- Inicializace toolkitu musí být volána před ostatními funkcemi GTK+.
- `gtk_init()` odebere z argumentů přepínače určené pro GTK+, např. `--display`, `--sync`, `--gtk-debug`.
- Vytvoření toplevel okna se provede funkcí `gtk_window_new()`, možné hodnoty parametru jsou `GTK_WINDOW_TOPLEVEL` (okno bude plně ovladatelné window managerem) a `GTK_WINDOW_POPUP` (není spravované window managerem).
- Připojený handler signálu se vykoná, když přijde událost (`delete_event` při zavření okna window managerem) nebo se s widgetem něco zajímavého stane a widget vygeneruje signál (`clicked` při stisku tlačítka, tomu ještě typicky předchází signály `enter`, `pressed` a následují ho `released`, `leave`).
- Funkce `gtk_container_add()` do okna přidá tlačítko, které bylo vytvořené voláním `gtk_button_new_with_label()`.
- Každý widget je nutné zobrazit funkcí `gtk_widget_show()`, widget včetně všech potomků lze najednou zobrazit pomocí `gtk_widget_show_all()`.
- Funkce `gtk_main()` je hlavní smyčka pro zpracování událostí, po jejím ukončení (vyvolaném `gtk_main_quit()`) končí program.
- V GTK+ se hodně používají přetypovací makra, která za běhu kontrolují, zda lze přetypování provést (obdoba `dynamic_cast` v C++).

Hello World (2)

- handler pro událost (speciální typ signálu)

```
gint delete_event(GtkWidget *widget, GdkEvent *event,  
                 gpointer data)  
{  
    g_print("delete event occurred\n");  
    return TRUE;  
}
```

- handler pro obecný signál

```
static void destroy(GtkWidget *widget, gpointer data)  
{  
    gtk_main_quit();  
}
```

- Pro logické hodnoty se používají konstanty FALSE a TRUE.
- Každý handler signálu má jako první parametr ukazatel na objekt, který vyvolal signál. Poslední parametr je obecný ukazatel nastavovaný při volání `g_signal_connect()`.
- Některé signály mohou mít ještě další parametry nebo návratovou hodnotu.
- Při nastavení handleru pomocí `g_signal_connect_swapped()` se vymění význam prvního a posledního parametru. To se využije, když jako handler použijeme nějakou funkci pracující s jiným objektem, než který vyvolal signál. V našem příkladu takto voláme `gtk_widget_destroy()` jako reakci na stisk tlačítka.
- Handlers pro události mají navíc ukazatel na strukturu popisující událost a vrací TRUE, jestliže událost byla obsloužena a FALSE, jestliže obsluha nebyla dokončena a událost se má dál propagovat (od widgetu, kde nastala, směrem k jeho rodičům).
- V našem příkladu by vrácení FALSE z `delete_event()` vyvolalo standardní obsluhu, tj. zrušení okna. To není způsobeno propagací události, ale tím, že když zpracování `delete_event` vrátí FALSE, `gtk_main()` widget automaticky zruší.
- Nestačí zrušení hlavního okna programu, je třeba explicitně ukončit cyklus zpracování událostí funkcí `gtk_main_quit()`.
- Místo definice handleru volajícího `gtk_main_quit()` je možné rovnou tuto funkci připojit jako signál:

```
g_signal_connect(G_OBJECT(window), "destroy",  
                G_CALLBACK(gtk_main_quit), NULL);
```

Slide 22

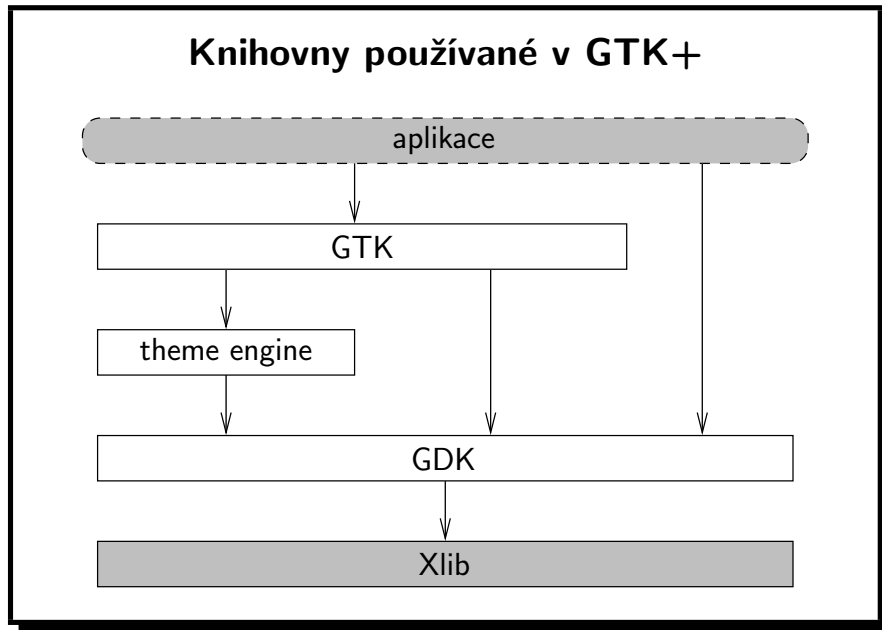
Překlad a linkování programu

- `#include <gtk/gtk.h>` ... hlavní hlavičkový soubor pro GTK+, zajistí načtení dalších souborů: `gdk/gdk.h`, `glib.h` a hlaviček pro jednotlivé widgety, např. `gtk/gtkwidget.h`, `gtk/gtkbutton.h`
- `pkg-config --cflags gtk+-2.0` ... vypíše cesty k hlavičkovým souborům GTK+ ve formě přepínačů kompilátoru `-Idir`
- `gtk-config --libs gtk+-2.0` ... vypíše jména potřebných knihoven a cesty k nim ve formě parametrů linkeru `-Ldir` a `-llib`
- překlad a slinkování programu:

```
gcc -o hello `pkg-config --cflags --libs gtk+-2.0` hello.c
```

- Místo volání `pkg-config` lze samozřejmě příslušné direktivy vložit přímo do volání kompilátoru, nebo si výstup `pkg-config` uložit do proměnné a tu pak používat.

Slide 23



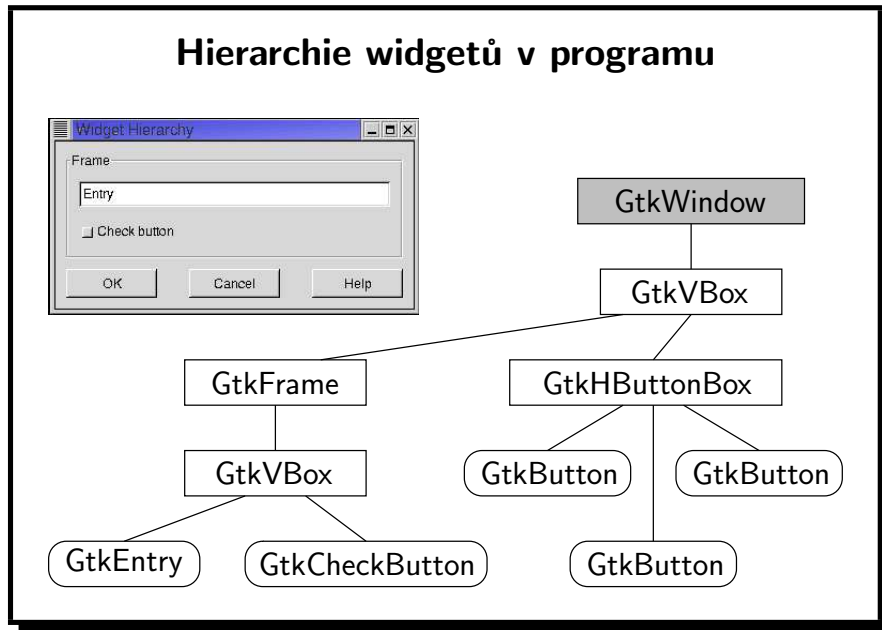
- Zde jsou znázorněny nejdůležitější knihovny a jejich vzájemné vztahy.

Knihovny používané v GTK+ (2)

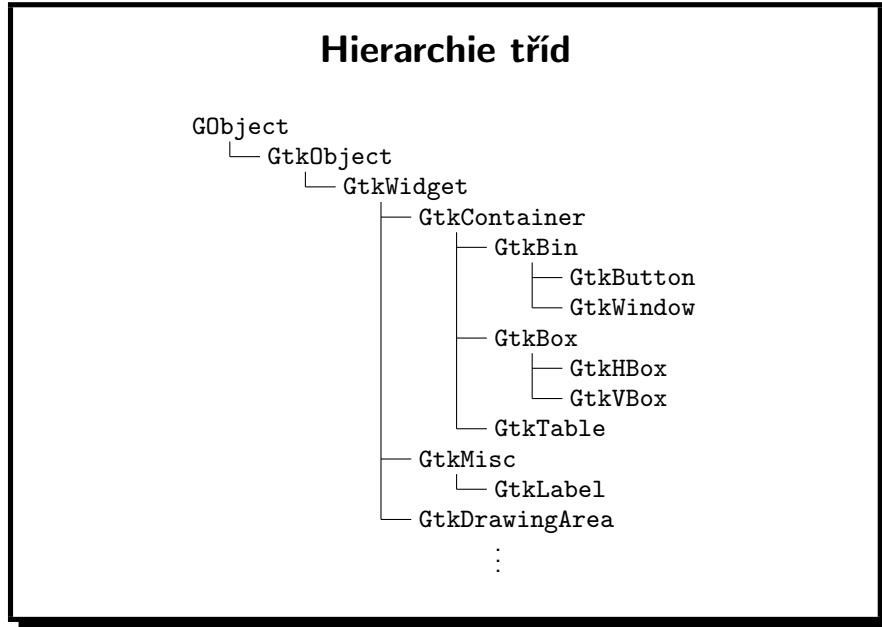
- **GTK+** (-lgtk-x11-2.0) ... vlastní toolkit GTK+
- **GDK** (-lgdk-x11-2.0) ... wrapper nad Xlib
- **GObject** (-lgobject-2.0) ... generický typový a objektový systém, signály
- **GLib** (-glib-2.0) ... pomocné funkce: řetězce, seznamy, stromy, hašovací tabulky, alokace paměti, obecná smyčka pro ošetření událostí, atd.
- **Pango** (-lpango-1.0) ... zobrazování textu
- **theme engines** ... knihovny definující grafickou podobu jednotlivých widgetů; dají se nahrávat za běhu a tím dynamicky měnit vzhled programu

- *GTK+* ... jádro GTK+ obalující generické objekty z GObject, kód pro jednotlivé widgety
- *GDK* ... obaluje jednotlivé funkce Xlib, něco z Xlib skrývá nebo zjednodušuje; pro port GTK+ do jiného grafického prostředí stačí portovat GDK; je portována pro MS Windows a linuxový framebuffer
- *GObject* ... GLib Object System, implementace objektů v C, včetně dědičnosti a virtuálních metod (jsou realizované pomocí signálů)
- *GLib* ... lze použít i samostatně
- další knihovny (vybrané):
 - *ATK* (-latk-1.0) ... Accessibility Toolkit, generické rozhraní pro zpřístupnění GTK+ tělesně postiženým uživatelům
 - *GdkPixbuf* (-lgdk_pixbuf-2.0) ... načítání a ukládání obrázků z/do souboru, manipulace s obrazovými daty (změna velikosti, vykreslování)
 - *GModule* (-lmodule-2.0) ... dynamické nahrávání objektových modulů (plugins)
 - *Xlib* (-lX11) ... základní knihovna X klienta
 - *Xext* (-lXext) ... základní podpora rozšíření (extensions) X
 - *iconv* (-liconv) ... převod mezi znakovými sadami

Slide 25



- Program má tolik stromů widgetů, kolik má toplevel oken.
- Každé toplevel okno (typicky `GtkWidget`), je kořenem jednoho stromu.
- Vnitřní uzly stromu (včetně kořene) jsou *kontejnery* – widgety odvozené od `GtkContainer`, které mohou obsahovat další widgety.
- V listech stromu mohou být i jiné widgety než kontejnery.
- V našem příkladu nejsou `GtkButton` a `GtkCheckButton` opravdovými listy stromu, protože to jsou ve skutečnosti kontejnery obsahující nápis typu `GtkLabel`.



- Na obrázku je pouze malá část stromu dědičnosti tříd v GTK+.
- Není vícenásobná dědičnost, jsou interfaces.
- `GObject` ... bazová třída objektové hierarchie
- `GtkObject` ... bazová třída hierarchie GTK+
- `GtkWidget` ... bazová třída pro všechny widgety
- `GtkContainer` ... widget, který může obsahovat další widgety (kontejner)
- `GtkBin` ... kontejner, který může obsahovat maximálně jeden widget
- `GtkButton` ... tlačítko, typicky obsahuje nápis typu `GtkLabel`
- `GtkWindow` ... toplevel okno
- `GtkBox` ... kontejner, který obsahuje řadu nebo sloupec widgetů
- `GtkHBox` ... box s vloženými widgety uspořádanými horizontálně v řadě
- `GtkVBox` ... box s vloženými widgety uspořádanými vertikálně ve sloupci
- `GtkTable` ... kontejner, který uspořádá vložené widgety do pravoúhlé sítě buněk
- `GtkMisc` ... abstraktní třída definující alignment (zarovnání relativně k rodiči) a padding (doplnění prostoru kolem widgetu)
- `GtkLabel` ... statický nápis
- `GtkDrawingArea` ... abstraktní třída pro definici uživatelských widgetů

Vytvoření a zobrazení widgetu

```
GtkWidget *gtk_typ_widgetu_new(parametry);
```

- Vytvoří a inicializuje widget, ale zatím mu nevytvoří X okno.

```
void gtk_widget_realize(GtkWidget *widget);
```

- Vytvoří k widgetu příslušné X okno.

```
void gtk_widget_show(GtkWidget *widget);
```

- Zobrazí widget.
- Aby byl widget skutečně vidět na obrazovce, musí být zobrazení všichni jeho předkové v hierarchii widgetů.
- Pokud widget nebyl předtím realizován, realizuje ho.

```
void gtk_widget_show_all(GtkWidget *widget);
```

- Zobrazí widget a všechny jeho potomky.

- Obvykle se po vytvoření widgetu volá pouze jediná funkce, a to `gtk_widget_show()`, resp. `gtk_widget_show_all()`.
- Někdy můžeme potřebovat použít odkaz na X okno dříve, než zobrazíme widget. V takovém případě je třeba zavolat `gtk_widget_realize()`.
- Ještě existuje funkce `gtk_widget_map()`, která je volaná z `gtk_widget_show()` a provede namapování X okna widgetu. X okno vytvořené pomocí `gtk_widget_realize()` není hned vidět. Aby se objevilo na obrazovce, je potřeba ho *namapovat*.
- Widget může být namapován, pouze když byl zobrazen pomocí `gtk_widget_show()`. Protože namapování se provede automaticky v reakci na `gtk_widget_show()`, má smysl ho volat pouze po předchozím `gtk_widget_unmap()`.

Schování a zrušení widgetu

```
void gtk_widget_hide(GtkWidget *widget);
```

- Schová widget.
- Dvojici volání `gtk_widget_show()` a `gtk_widget_hide()` lze libovolně opakovat.

```
void gtk_widget_destroy(GtkWidget *widget);
```

- Zruší widget (včetně příslušného X okna).

- Funkce `gtk_widget_hide()` odmapuje X okno pomocí `gtk_widget_unmap()`.
- Volání `gtk_widget_unmap()` pouze schová X okno widgetu a tím vznikne prázdné místo. Funkce `gtk_widget_hide()` navíc přepočítá rozložení ostatních widgetů tak, že výsledek vypadá, jako by schovaný widget vůbec neexistoval.
- Ještě existuje funkce `gtk_widget_unrealize()`, která zruší X okno widgetu.

Nastavení obsluhy signálů

```
gulong g_signal_connect(GObject *object,
                        const gchar *name, GCallback func,
                        gpointer func_data);
```

- Nastaví obsluhu signálu name emitovaného objektem object tak, že se bude volat funkce func, jako první parametr dostane object a jako poslední parametr data.
- Vrací identifikátor připojení signálu.

```
gulong g_signal_connect_swapped(GObject *object,
                                const gchar *name, GCallback func,
                                GObject *slot_object);
```

- Nastaví obsluhu signálu, ale handler bude dostávat slot_object jako první a object jako poslední parametr.

- Handlers připojené ke stejnému objektu a signálu se volají v pořadí, v jakém byly připojeny.
- Varianta `g_signal_connect_swapped()` se používá, když chceme jako reakci na signál zavolat pro nějaký jiný objekt (widget) funkci GTK+, která má jako parametr ukazatel na objekt, např. `gtk_widget_destroy()` (jako v našem „Hello World“), `gtk_widget_show()`, `gtk_widget_hide()`, apod.
- Často se využívá toho, že v C se o parametry volání funkce stará volající. Proto můžeme jako handler použít i funkci, která má méně parametrů, než kolik jí GTK+ předává (nadbytečné parametry se ignorují).
- Měli bychom vždy dodržet typ návratové hodnoty handleru.
- V referenční dokumentaci GTK+ je u každé třídy popsáno, jaké signály jsou definovány pro objekty této třídy. Objekt může používat i signály kteréhokoli předka v hierarchii dědičnosti tříd.
- Třídy mají k signálům definovanou standardní obsluhu (default handler).
- U každého signálu je definováno, jakého typu má být handler, tj. kolik a jakého typu má mít parametrů a jaký je typ návratové hodnoty, pokud má signál návratovou hodnotu. Dodržování správných typů handlerů je ponecháno na programátorovi, GTK+ to nekontroluje.
- Příklad signálů s návratovou hodnotou jsou signály emitované na základě událostí. Vracejí boolovskou hodnotu, která říká, zda byla událost ošetřena nebo se pro ni mají volat další handlers a případně se má propagovat do rodičovského widgetu.

Nastavení obsluhy signálů (2)

```
gulong g_signal_connect_after(GObject *object,  
    const gchar *name, GCallback func,  
    gpointer func_data);
```

- Obdoba `g_signal_connect()`, ale handler se volá až po standardním handleru.

```
void g_signal_handler_disconnect(gpointer *object,  
    gulong handler_id);
```

- Odpojí handler zadaný pomocí `handler_id` (návrátová hodnota z předchozího `gtk_signal_connect()`) od objektu `object`.

Slide 30

- Existují i další funkce, např.:
 - `g_signal_emit_by_name()` ... vygenerování signálu zadaného jména
 - `g_signal_emit()` ... vygenerování signálu se zadaným ID
 - `g_signal_stop_emission_by_name()` ... přerušování zpracování signálu, zbývající handlery nebudou volány
 - `gtk_signal_add_emission_hook()` ... nastavení funkce, která se spustí, pokud je určitý signál vyvolán pro libovolný objekt
- Signály jsou rozděleny do dvou skupin podle toho, kdy se volá standardní handler: `G_SIGNAL_RUN_FIRST` a `G_SIGNAL_RUN_LAST`.
- Postup obsluhy signálu:
 1. standardní handler, pokud je signál `G_SIGNAL_RUN_FIRST`
 2. globální handlery (emission hooks)
 3. handlery připojené pomocí `g_signal_connect()` a `g_signal_connect_swapped()`
 4. standardní handler, pokud je signál `G_SIGNAL_RUN_LAST`
 5. handlery připojené pomocí `g_signal_connect_after()`

Slide 31

Cyklus zpracování událostí

```
void gtk_main(void);
```

- Spustí cyklus zpracování událostí.
- Tato funkce se obvykle volá po vytvoření hlavního okna programu a jejím ukončením končí program.
- Lze volat rekurzivně, např. z handleru signálu.

```
void gtk_main_quit(void);
```

- Ukončí nejnvnitřnější `gtk_main()`.

```
gboolean gtk_main_iteration(void);
```

- Čeká na událost a provede jednu iteraci smyčky zpracování událostí.

```
gint gtk_events_pending(void);
```

- Otestuje, zda existují nezpracované události.

- Rekurzivní volání `gtk_main()` se používá např. při implementaci modálních dialogů. Nejdříve se vytvoří dialog, nastaví se modalita pomocí `gtk_window_set_modal()`, pak se zavolá `gtk_main()`. Dialog má v handleru pro signál `destroy` volání `gtk_main_quit`, což způsobí, že se `gtk_main()` vrátí po zavření dialogu. My pak můžeme zpracovat nastavení provedená uživatelem v dialogu.
- Funkce `gtk_main_iteration()` vrací `TRUE`, pokud pro nejnvnitřnější `gtk_main()` bylo zavoláno `gtk_main_quit()`.

Správa paměti

- GTK+ udržuje u každého objektu počet referencí na něj.

```
void g_object_ref(gpointer *object);
```

- Přidá referenci na objekt.
- Voláno kontejnerem na vložené objekty.

```
void g_object_unref(gpointer *object);
```

- Sníží počet referencí na objekt, když počet klesne na 0, zruší objekt.
- Voláno kontejnerem na vložený objekt při jeho odebrání z kontejneru.

```
void gtk_object_sink(GtkObject *object);
```

- Odstraní počáteční referenci na objekt, ale jen jednou.
- Voláno kontejnerem na vložené objekty.

- Obvykle není potřeba se příliš starat o manipulaci s počtem odkazů na objekty ani o rušení objektů.
- Zrušením kontejnerového widgetu se zruší i všechny do něho vložené widgety.
- I když je widget zrušený, paměť je uvolněna, teprve až počet odkazů na objekt klesne na 0. Lze tedy používat ukazatel na zrušený widget, pokud jsme pro tento ukazatel předtím zvýšili počet referencí.
- Objekty jsou vytvářeny s počtem odkazů 1.
- Funkce `gtk_widget_destroy()` (zrušení widgetu) provede `g_object_unref()`.
- Abychom nemuseli volat `g_object_unref()` na každý widget, který vložíme do nějakého kontejneru, jsou objekty vytvářeny tzv. „plovoucí“ (floating). Kontejner nejdříve přidá na objekt referenci a pak ho „potopí“ (odebere první referenci) pomocí funkce `gtk_object_sink()`. Druhé a další volání `gtk_object_sink()` na stejný objekt nemají žádný efekt.
- Když chceme odebrat widget z kontejneru, ale nezrušit ho, musíme nejdříve přidat referenci, tj. volat `g_object_ref()` před `gtk_container_remove()`.
- Někdy nechceme, aby se některý pointer počítal do počtu referencí a nebránil tak zrušení objektu, např. v nějaké cyklické struktuře, kde by počet referencí byl vždy aspoň 1 a struktura by se nezrušila po zrušení posledního odkazu na ni. Pak je možné nastavit pointer na objekt bez volání `g_object_ref()`, jenže se nedozvíme, kdy se objekt zruší a zbyde nám neplatný ukazatel. Lepší je pomocí `g_object_weakref()` zaregistrovat funkci, která se zavolá při zrušení objektu. Tuto funkci můžeme opět odregistrovat voláním `g_object_weakunref()`.

Kontejnery

- **obecný kontejner** (`GtkContainer`) ... rodičovská třída pro všechny kontejnery
- **bin** (`GtkBin`) ... např. `GtkButton`, `GtkWindow`; obsahuje jediný synovský widget, ukazatel na něj je `GtkBin.child`
- **box** (`GtkBox`) ... obsahuje jednu posloupnost widgetů řazenou od začátku a jednu řazenou od konce boxu; řadí widgety buď vedle sebe (`GtkHBox`) nebo pod sebe (`GtkVBox`)
- **tabulka** (`GtkTable`) ... definuje řádky a sloupce, k nimž se přichytí okraje synovských widgetů
- Kontejnery se mohou vnořovat.

- Další kontejnery:
 - `GtkAlignment` ... potomek `GtkBin`, lze definovat relativní velikost a pozici vloženého widgetu v rámci kontejneru
 - `GtkFixed` ... může obsahovat více widgetů, pro každý synovský widget se zadává pozice v pixelech
- Nejvíce se pro definování vzhledu rozhraní používají boxy a tabulky. V nich se definují vzájemné polohy jednotlivých widgetů. Přesné polohy a velikosti widgetů se alokují dynamicky.
- Toplevel okno (`GtkWindow`) je odvozeno od `GtkBin` a může obsahovat pouze jeden synovský widget. Proto se do okna obvykle vloží tabulka nebo box a teprve do něho se vkládají další widgety.

Společné funkce pro kontejnery

```
void gtk_container_add(GtkContainer *container,
                       GtkWidget *widget);
```

- Vloží widget do kontejneru.

```
void gtk_container_remove(GtkContainer *container,
                           GtkWidget *widget);
```

- Odebere widget z kontejneru.

```
void gtk_container_set_border_width(GtkContainer *container,
                                     guint border_width);
```

- Nastaví velikost okraje, do kterého nezasahují synovské widgety.

```
void gtk_widget_set_size_request(GtkWidget *widget,
                                  gint width, gint height);
```

- Nastaví minimální velikost widgetu.

- Pro různé typy kontejnerů jsou obvykle definovány speciální funkce pro vkládání widgetů, např. `gtk_box_pack_start()`, `gtk_table_attach()`. Obecná metoda pro vkládání `gtk_container_add()` je předefinována tak, že má význam jedné z těchto speciálních funkcí.
- Když odebereme widget z kontejneru a na widget neexistuje další reference, bude zrušen.
- Funkce `gtk_widget_set_size_request()` je definována pro všechny widgety, nejen pro kontejnery. Každý widget má standardní minimální velikost. Ta je typicky taková, aby šel widget rozumně zobrazit (např. tlačítko je tak velké, aby se zobrazil celý nápis na něm). Kontejnery mohou widgety při umísťování zvětšit nad minimální velikost, ale obvykle je nezmenšují. Funkce `gtk_widget_set_size_request()` umožňuje nastavit jinou než standardní velikost widgetu, např. kdybychom chtěli kolem textu na tlačítku přidat místo – konkrétně toto se lépe vyřeší pomocí `gtk_misc_set_padding(GTK_MISC(GTK_BIN(button)->child), 10, 5);`
- Pro nastavení počáteční velikosti okna je lepší použít `gtk_window_set_default_size()`, protože uživatel pak může okno zmenšit.

Slide 35

Velikost widgetů

- Nejprve se sbírají požadavky na velikost jednotlivých widgetů. Minimální velikost, kterou požaduje widget, vrátí
`void gtk_widget_size_request(GtkWidget *widget,
GtkRequisition *requisition);`
Pro kontejnery tato metoda nejprve zjistí požadavky všech potomků a pak spočítá požadovanou velikost celého kontejneru.
- Toplevel okno nastaví svoji velikost (v součinnosti s window managerem).
- Dále se postupuje rekurzivně směrem k listům. Velikost widgetu se nastaví pomocí
`void gtk_widget_size_allocate(GtkWidget *widget,
GtkAllocation *allocation);`
Kontejner následně přidělené místo rozdělí mezi své potomky.

- Nastavování velikosti obvykle probíhá zcela automaticky.
- Způsob, jak kontejner poskládá požadavky svých potomků ve stromu widgetů do vlastní požadované velikosti, a způsob distribuce přiděleného místa mezi potomky závisí na typu kontejneru.

Slide 36

Boxy

```
GtkWidget* gtk_hbox_new(gboolean homogeneous,  
                        gint spacing);
```

```
GtkWidget* gtk_vbox_new(gboolean homogeneous,  
                        gint spacing);
```

- vytvoření prázdného boxu
- *homogeneous* ... všechny synovské widgety stejně velké
- *spacing* ... mezery mezi syny

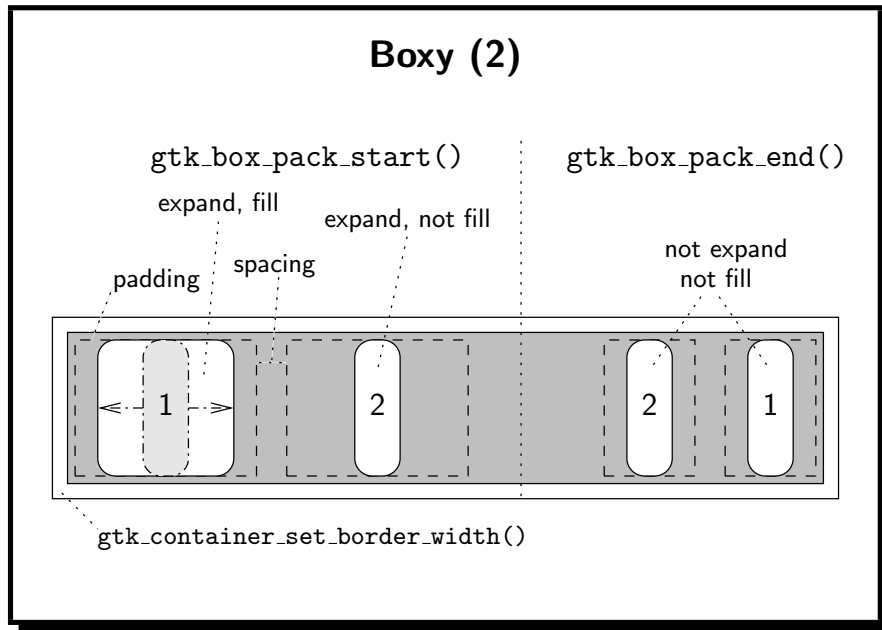
```
void gtk_box_pack_start(GtkBox *box, GtkWidget *child,  
                       gboolean expand, gboolean fill, guint padding);
```

```
void gtk_box_pack_end(GtkBox *box, GtkWidget *child,  
                      gboolean expand, gboolean fill, guint padding);
```

- vložení widgetu do boxu

- Všechny widgety vložené pomocí `gtk_box_pack_start()` jsou zobrazeny blíže k začátku (vlevo nebo nahoře) boxu než widgety vložené pomocí `gtk_box_pack_end()`.
- Widgety vložené funkcí `gtk_box_pack_start()` jsou uspořádány zleva doprava (shora dolů) v pořadí, v jakém byly vkládány.
- Widgety vložené funkcí `gtk_box_pack_end()` jsou uspořádány zprava doleva (zdola nahoru) v pořadí, v jakém byly vkládány.

Slide 37



- Plocha boxu rozdělená mezi syny je celý box zmenšený na všech stranách o okraj nastavený pomocí `gtk_container_set_border_width()`.
- `expand ...` řídí, zda widget dostane část nadbytečného místa v boxu
- `fill ...` řídí, zda widget vyplní nadbytečné místo (`TRUE`), nebo zda bude mít minimální velikost a místo zůstane po stranách (`FALSE`)
- `spacing ...` mezera mezi syny
- `padding ...` přidané místo po stranách každého syna
- Synové hboxu zabírají vždy plnou výšku boxu (zmenšenou o okraje). Obdobně synové vboxu zabírají plnou šířku boxu.
- Obrázek je trochu nepřesný: widgety s nastaveným `expand` vždy dostanou veškeré nadbytečné místo v boxu. Proto v boxu s aspoň jedním `expand` widgetem není mezera mezi posledním widgetem vloženým pomocí `gtk_box_pack_start()` a posledním widgetem vloženým pomocí `gtk_box_pack_end()`.

Tabulky

```
GtkWidget* gtk_table_new(guint rows, guint columns,  
                          gboolean homogeneous);
```

- Vytvoří tabulku.
- *homogeneous* ... všechna políčka stejně velká

```
void gtk_table_attach_defaults(  
    GtkTable *table, GtkWidget *widget,  
    guint left_attach, guint right_attach,  
    guint top_attach, guint bottom_attach);
```

- Vloží widget do tabulky. Levý horní roh widgetu bude ve sloupci *left_attach* a řádku *top_attach*, pravý dolní roh ve sloupci *right_attach* a řádku *bottom_attach*.

Slide 38

```
void gtk_table_attach(GtkTable *table, GtkWidget *child,  
    guint left_attach, guint right_attach,  
    guint top_attach, guint bottom_attach,  
    GtkAttachOptions xoptions, GtkAttachOptions yoptions,  
    guint xpadding, guint ypadding);
```

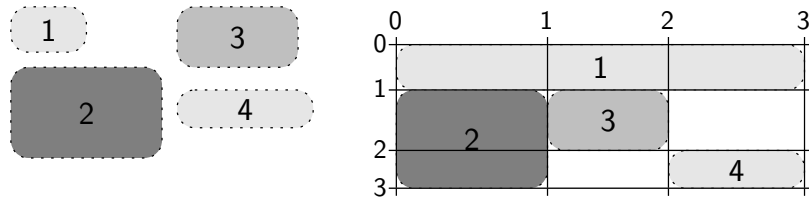
- Umožňuje specifikovat zvlášť v *x*-ovém a *y*-ovém směru přidělování nadbytečného místa (*xoptions* a *yoptions* obsahující konstanty `GTK_EXPAND` a `GTK_FILL` s podobným významem jako parametry *expand* a *fill* při vkládání do boxu) a přidávání místa kolem každého synovského widgetu.
- V parametrech *xoptions* a *yoptions* může být také konstanta `GTK_SHRINK`, která povolí zmenšení widgetu, pokud pro něj v tabulce není dost místa.
- Volání `gtk_table_attach_defaults()` je ekvivalentní

```
gtk_table_attach(table, widget, l_att, r_att, t_att, b_att,  
                GTK_EXPAND | GTK_FILL, GTK_EXPAND | GTK_FILL, 0, 0);
```

Tabulky (2)

```
table = gtk_table_new(3, 3, FALSE);  
gtk_table_attach_defaults(table, widget1, 0, 3, 0, 1);  
gtk_table_attach_defaults(table, widget2, 0, 1, 1, 3);  
gtk_table_attach_defaults(table, widget3, 1, 2, 1, 2);  
gtk_table_attach_defaults(table, widget4, 2, 3, 2, 3);
```

Slide 39



- Šířky a výšky políček tabulky nejsou pevné, ale pružně se mění podle požadavků na minimální velikosti jednotlivých vložených widgetů. Lze si to představit tak, že sloupce a řádky tabulky jsou volně pohyblivé tyče (samozřejmě při zachování pořadí, tj. nemohou se např. dva sloupce prohodit) a k nim se připevňují okraje widgetů. Výsledné vzdálenosti mezi sloupci a řádky tabulky budou takové, aby se widgety pokud možno nemusely zmenšovat.

Stavy widgetu

- Widget, který má **focus**, bude dostávat vstup z klávesnice, je orámován tenkou přerušovanou čarou. Přepínání: klávesy Tab a Shift+Tab.

GTK_WIDGET_SET_FLAGS(*wid*, *flag*)

GTK_WIDGET_UNSET_FLAGS(*wid*, *flag*)

- Makra pro nastavení a zrušení příznakových bitů

```
void gtk_widget_set_sensitive(GtkWidget *widget,  
                               gboolean sensitive);
```

- Nastaví, zda widget reaguje na podněty uživatele.
- Když je nastaveno na FALSE, widget to vizuálně indikuje (tlačítko nebo položka menu bude mít šedý text místo černého).

- Defaultní widget (typicky tlačítko) je ten, který zareaguje na Enter, i když nemá focus. Vizuálně je indikován orámováním tenkou černou čarou. Musí mít nastaven flag `GTK_CAN_DEFAULT` a pak může získat default pomocí `gtk_widget_grab_default()`.
- Pro testování stavu slouží makra: `GTK_WIDGET_IS_SENSITIVE`, `GTK_WIDGET_HAS_FOCUS`, `GTK_WIDGET_HAS_DEFAULT`, apod.

Slide 41

Widety bez X oken

| | | |
|----------------|-------------------|----------------|
| GtkAlignment | GtkLabel | GtkTable |
| GtkArrow | GtkMenuItem | GtkToolbar |
| GtkBin | GtkNotebook | GtkAspectFrame |
| GtkBox | GtkPaned | GtkFrame |
| GtkButton | GtkRadioButton | GtkVBox |
| GtkCheckButton | GtkRange | GtkHBox |
| GtkFixed | GtkScrolledWindow | GtkVSeparator |
| GtkImage | GtkSeparator | GtkHSeparator |

- Test, zdá má widget X okno: makro **GTK_WIDGET_NO_WINDOW**(*widget*)

- Tyto widety nemají vlastní X okno.
- Aby se mohl widget vykreslit, potřebuje X okno. Tyto widety kreslí do okna rodiče.
- Widety bez oken nemohou dostávat události. Pokud je potřeba pro ně zpracovávat události, dají se vložit do kontejneru `GtkEventBox`.
- Seznam widgetů bez oken je opsaný z GTK+ 2.0 Tutorial.

Tlačítko (GtkButton)

```
GtkWidget* gtk_button_new_with_mnemonic(const gchar *label);
```

- Vytvoří tlačítko se zadaným nápisem. Znaky, před kterými je '_', jsou podtržené, první z nich je akcelerátor.

```
GtkWidget* gtk_button_new(void);
```

- Vytvoří prázdné tlačítko, lze do něj přidat libovolný widget (tlačítko je kontejner, potomek GtkBin).

Tlačítko vygeneruje signál "clicked", když ho uživatel stiskne.

```
GTK_WIDGET_SET_FLAGS(button, GTK_CAN_DEFAULT);  
gtk_widget_grab_default(button);
```

- Tlačítko dostane výraznější orámování a uživatel ho může stisknout klávesou Enter, i když tlačítko nemá focus.

Slide 42

- Funkce `gtk_button_new()` se používá, když na tlačítku nemá být jen text, ale např. obrázek.
- Funkce `gtk_button_new_with_mnemonic()` nejprve vytvoří prázdné tlačítko pomocí volání `gtk_button_new()` a pak do něj vloží nápis (typu `GtkLabel`). Znak '_' musíme zdvojit, pokud jím neoznačujeme podtržení následujícího znaku.
- Funkcí `gtk_button_new_with_label()` dostaneme label tlačítka bez akcelerátoru a bez podtržení.
- Další signály: "enter", "pressed", "released", "leave"

Tlačítka GtkToggleButton, GtkCheckButton

- Mohou být v jednom ze dvou stavů, přepínají se kliknutím.
- Při změně stavu generují signál "toggled".
- Liší se vzhledem: toggle button vypadá jako normální tlačítko, check button je malé tlačítko s nápisem vedle.

```
GtkWidget* gtk_toggle_button_new_with_label(const gchar *lbl);
```

```
GtkWidget* gtk_check_button_new_with_label(const gchar *lbl);
```

- Vytvoří tlačítko příslušného typu s nápisem.

```
gboolean gtk_toggle_button_get_active(GtkToggleButton *btn);
```

- Zjistí stav tlačítka.

```
void gtk_toggle_button_set_active(GtkToggleButton *btn,  
                                  gboolean is_active);
```

- Nastaví stav tlačítka.

- Alternativně lze použít funkce `..._with_mnemonic`.
- Přepínací tlačítka indikují vizuálně svůj stav (stisknuto/nestisknuto).
- Třída `GtkCheckButton` je potomkem třídy `GtkToggleButton`.
- Jako u obyčejného tlačítka existují i funkce `gtk_toggle_button_new()` a `gtk_check_button_new()`.

Slide 44

Tlačítka GtkRadioButton

- Obdoba tlačítek typu check button, ale vždy je aktivní právě jedno ze skupiny několika tlačítek typu radio button.
- Skupina tlačítek je svázána pomocí seznamu typu GList *.
- Vytvoření skupiny:

```
btn1 = gtk_radio_button_new_with_label(NULL, "Button 1");
btn2 = gtk_radio_button_new_with_label(
    gtk_radio_button_group(GTK_RADIO_BUTTON(btn1)),
    "Button 2");
btn3 = gtk_radio_button_new_with_label(
    gtk_radio_button_group(GTK_RADIO_BUTTON(btn2)),
    "Button 3");
```

- Po vytvoření skupiny je aktivní první tlačítko.
- Lze i `..._with_mnemonic()`.
- Alternativně lze pro vytvoření tlačítka použít:

```
btn2 = gtk_radio_button_new_with_label_from_widget(GTK_RADIO_BUTTON(btn1),
    "Button 2");
```

Slide 45

Adjustment (GtkAdjustment)

- Adjustment není widget, ale objekt, který uchovává informaci pro scrollování.
- Položky struktury GtkAdjustment, všechny jsou typu gdouble:
 - *lower* ... minimální hodnota
 - *upper* ... maximální hodnota souřadnice v posouvaném widgetu; maximální posun je `upper - page_size`
 - *value* ... aktuální hodnota
 - *step_increment* ... menší hodnota posunu, scrollbar ji použije při kliknutí na koncovou šipku (posun o řádek)
 - *page_increment* ... větší hodnota posunu, scrollbar ji použije při kliknutí mezi indikátor pozice a šipku (posun o stránku)
 - *page_size* ... viditelná velikost stránky v posouvaném widgetu

- Pokud chceme scrollovat v celém rozsahu `lower` až `upper`, můžeme nastavit `page_size` na nulu.
- Když chceme scrollovat po stránkách podle velikosti okna, nastavíme `page_increment` i `page_size` na velikost okna.

Slide 46

Adjustment (2)

- Při typickém použití je sdílen scrollbar a jiným widgetem, který je ovládán (posouván) pomocí scrollbaru.
- Změny nastavení se provádí přiřazením do jednotlivých položek.
- Po změně hodnoty (`value`) je třeba zavolat `void gtk_adjustment_value_changed(GtkAdjustment *adj);` Tím se vygeneruje signál "value_changed" a připojené widgety zaregistrují změnu.
- Po změně jiné položky se pošle signál "changed" pomocí `void gtk_adjustment_changed(GtkAdjustment *adj);`

- Jeden adjustment může být sdílen i jiným způsobem než jen mezi scrollbar a jím ovládaným widgetem, např. dva scrollbary mohou být navzájem svázané pomocí jednoho adjustmentu nebo jeden scrollbar může řídit posun ve dvou oknech.
- Widgety, které počítají s tím, že mohou mít přiřazen adjustment (jako `GtkScrollbar`, `GtkScale` a `GtkScrolledWindow`) automaticky reagují na signály "value_changed" a "changed".

Slide 47

Posouvátka (GtkScrollbar, GtkScale)

- Widgety pro nastavení pozice
- Potomci společného předka GtkRange
- Oba typy existují v horizontální (GtkHScrollbar a GtkHScale) a vertikální (GtkVScrollbar a GtkVScale) variantě.
- Mají připojený adjustment, který se zadává při vytvoření widgetu. Rozsah a kroky posunu se nastaví podle adjustmentu. Posouvátko řídí hodnotu adjustmentu.
- Liší se vzhledem.
- Scrollbar se obvykle používá pro posun obsahu jiného widgetu, jako je okno s tabulkou nebo textovým souborem.
- Scale je určen pro zadávání numerické hodnoty z nějakého intervalu.

- Scale nemá šipky na koncích a volitelně může zobrazovat aktuálně nastavenou hodnotu.

Statický text (GtkLabel)

- Krátký text, typické použití jsou popisky položek v dialogích.

```
GtkWidget* gtk_label_new(const gchar *str);
```

```
void gtk_label_set_text_with_mnemonic(GtkLabel *l, gchar *s);
```

- Zadání textu při vytvoření widgetu nebo později

```
void gtk_label_set_line_wrap(GtkLabel *l, gboolean wrap);
```

- Způsob ukončení řádků: znakem '\n' nebo automaticky

```
void gtk_label_set_justify(GtkLabel *l, GtkJustification jtyp);
```

- Zarovnání víceřádkového textu

```
void gtk_label_set_mnemonic_widget(GtkLabel *l, GtkWidget *w);
```

- Widget aktivovaný stiskem akcelerátoru

Slide 48

- Statický text nemá vlastní X okno.
- Text může být i víceřádkový, funkcí `gtk_label_set_line_wrap()` lze nastavit automatické ukončování řádků.
- Při automatickém ukončování řádků znamená znak '\n' konec odstavce.
- *Justification* definuje vzájemnou polohu řádek víceřádkového textu.
- *Alignment* (`gtk_misc_set_alignment()`) je posun celého widgetu v rámci přiděleného místa, pokud je toto místo větší než přirozená velikost widgetu.

Šipky (GtkArrow)

```
GtkWidget* gtk_arrow_new(GtkArrowType arrow_type,  
                          GtkShadowType shadow_type);
```

- *arrow_type* ... směr, kterým šipka ukazuje: GTK_ARROW_UP, GTK_ARROW_DOWN, GTK_ARROW_LEFT, GTK_ARROW_RIGHT
- *shadow_type*:
 - GTK_SHADOW_IN ... snížený reliéf
 - GTK_SHADOW_OUT ... zvýšený reliéf
 - GTK_SHADOW_ETCHED_IN ... snížený reliéf obrysu
 - GTK_SHADOW_ETCHED_OUT ... zvýšený reliéf obrysu

Slide 49

- Šipky nemají vlastní X okna.
- Default theme v GTK+ 2 nepoužívá reliéf, zobrazuje šipky jako černé trojúhelníčky.

Tooltipy (GtkTooltips)

```
GtkTooltips* gtk_tooltips_new(void);
```

- Vytvoří skupinu tooltipů.
- Pro celou skupinu najednou lze nastavovat, jestli se v ní obsažené tipy budou zobrazovat a jaká je prodleva před zobrazením tooltipu.

```
void gtk_tooltips_set_tip(GtkTooltips *tips, GtkWidget *widget,  
    const gchar *tip_text, const gchar *tip_private);
```

- Přidá tooltip s textem `tip_text` k widgetu `widget` a do skupiny `tips`.
- Text `tip_private` se nezobrazuje, ale dá se získat pomocí `gtk_tooltips_data_get()` a použít pro implementaci rozsáhlejší kontextově závislé nápovědy.

Slide 50

- Tooltip je krátká nápověda, která se zobrazí v rámečku u widgetu, pokud nad widgetem chvíli podržíme myš.
- Jedna skupina tooltipů může obsahovat tipy pro mnoho widgetů. Proto stačí funkci `gtk_tooltips_new()` použít pouze jednou.
- Položka `tip_private` může obsahovat klíč, podle kterého se vybere příslušný text ze souboru s nápovědou.
- Funkce `gtk_tooltips_data_get()` dostane jako argument ukazatel na widget a vrací strukturu `GtkTooltipsData` (`tooltips`, `widget`, `tip_text`, `tip_private`) pro tento widget.

Progress bar (GtkProgressBar)

- Ukazatel stavu nějaké déle trvající operace
- Stav progress baru se opakovaně aktualizuje funkcí

```
void gtk_progress_bar_set_fraction(GtkProgressBar *bar,  
                                   gdouble fraction);
```

Slide 51

- Používá adjustment.
- Může zobrazovat text (např. aktuální hodnotu nebo procenta).
- V „activity mode“ zobrazuje pouze pohybující se blok indikující, že se něco děje.

- *Activity mode* se používá, pokud je potřeba indikovat probíhající činnost, ale není žádný odhad, kolik ještě zbývá do konce.
- Activity mode se zapne prvním voláním

```
void gtk_progress_bar_pulse(GtkProgressBar *pbar);
```


Každé další volání způsobí posun o jeden krok.

Obrázky (GtkImage)

```
GtkWidget *gtk_image_new_from_file(const gchar file);
```

- Nahraje obrázek ze souboru a vytvoří widget. Jestliže se nepodaří načíst obrázek, použije standardní ikonu „broken image“.
- Když chceme reagovat na chyby při čtení obrázku, musíme nejprve vytvořit GtkPixbuf a teprve z něho GtkImage:

```
GError *err = NULL;  
if(!(pixbuf = gdk_pixbuf_new_from_file(fname, &err))) {  
    g_printerr("%s\ n", err->message); g_clear_error(&err);  
    return 1;  
}  
widget = gtk_image_new_from_pixbuf(pixbuf);  
g_object_unref(pixbuf);
```

Slide 52

- Widget GtkImage nemá vlastní X okno.
- Umí číst různé obrazové formáty.

Slide 53

Stavový řádek (GtkStatusbar)

- Udržuje zásobník textových zpráv a poslední z nich zobrazuje.

```
guint gtk_statusbar_get_context_id(GtkStatusbar *statusbar,  
                                   const gchar *context_desc);
```

- Vrátil identifikátor kontextu pro použití v dalších funkcích.

```
guint gtk_statusbar_push(GtkStatusbar *statusbar,  
                         guint context_id, const gchar *txt);
```

- Přidá zprávu do zásobníku a zobrazí ji.

```
void gtk_statusbar_pop(GtkStatusbar *sbar, guint context_id);
```

- Odebere poslední zprávu s daným kontextem.

```
void gtk_statusbar_remove(GtkStatusbar *statusbar,  
                          guint context_id, guint msg_id);
```

- Odebere konkrétní zprávu.

- Stavový řádek se obvykle umísťuje na spodní okraj hlavního okna aplikace.
- V zásobníku jsou zprávy v pořadí, jak byly vloženy. Naposled vložená nesmazaná zpráva je zobrazena.
- Zprávy mohou mít kontext, který se použije při výběru zprávy, která bude smazána – při mazání zprávy ze zásobníku se vybere poslední se zadaným kontextem.
- Pro důležité zprávy je lepší používat dialogy, protože změna ve stavovém řádku se dá snadno přehlédnout.

Slide 54

Editační řádek (GtkEntry)

- Potomek GtkEditable
- Jeden řádek textu, fungují v něm běžné editační klávesy.
- Dá se omezit maximální délka vloženého textu.
- Lze zakázat možnost editace pomocí `gtk_entry_set_editable()`.
- Místo textu je možné zobrazit hvězdičky (např. pro zadávání hesel).
- Má funkce na změnu textu (`gtk_entry_set_text()`, `gtk_editable_insert_text()`, `gtk_editable_delete_text()`).
- Obsah řádku vrátí funkce `gtk_entry_get_text()`.
- Automaticky podporuje označování textu a copy&paste.
- Umožňuje nastavit pozici v textu – `gtk_editable_set_position()`.
- Signál "changed" – uživatel provedl změnu, např. vložil znak.
- Signál "activate" – uživatel stiskl Enter.

- GtkEntry a GtkEditable definují řadu signálů, které se emitují při vložení "insert-text" nebo smazání textu "delete-text" nebo se jimi dá nastavovat pozice kurzoru "move-cursor", apod.

Slide 55

Spin button (GtkSpinButton)

- Odvozeno z GtkEntry
- Určeno k zadávání numerických hodnot, má připojený adjustment, ve kterém je povolený rozsah hodnot a aktuální hodnota – v tom se podobá GtkScale.
- Vypadá jako editační řádek a vedle něho dvě šipky.
- Hodnotu lze přímo napsat do řádku nebo zvětšovat a zmenšovat pomocí šipek.

- Dají se nastavovat parametry jako počet desetinným míst, wrapping (jestli po dosažení největší hodnoty se pokračuje od nejmenší) a akcelerace při delším stisku šipek.
- Hodnotu lze nastavit absolutně (funkce `gtk_spin_button_set_value()`) nebo relativně (funkce `gtk_spin_button_spin()`).

Slide 56

Editační řádek se seznamem (GtkCombo)

- Sestává z editačního řádku a rozbalovacího menu.
- Uživatel může buď přímo zadat text nebo si vybrat z menu.
- Vytvoření widgetu a nastavení obsahu menu:

```
GtkWidget *combo; GList *items = NULL;  
items = g_list_append(items, "Prvni");  
items = g_list_append(items, "Druhy");  
items = g_list_append(items, "Treti");  
combo = gtk_combo_new();  
gtk_combo_set_popdown_strings(GTK_COMBO(combo), items);
```

- Dá se zakázat měnit hodnotu v editačním řádku a dovolit pouze výběr z nabídky:

```
gtk_entry_set_editable(GTK_ENTRY(GTK_COMBO(item)->entry),  
FALSE);
```

- Příliš dlouhý seznam nabízených hodnot bude mít automaticky scrollbar.

Nabídka možností (GtkOptionMenu)

- Rozbalovací menu ze kterého si uživatel vybere jednu položku.
- Odpovídá combo boxu se zakázanou editací obsahu řádku, ale nemá scrollbar. Místo toho jsou na koncích zobrazené části seznamu rolovací šipky.

Slide 57

```
void gtk_option_menu_set_menu(GtkOptionMenu *option_menu,  
                               GtkWidget *menu);
```

- Nastaví menu s nabídkou hodnot.

```
void gtk_option_menu_set_history(GtkOptionMenu *option_menu,  
                                 guint index);
```

- Nastaví položku s daným indexem jako vybranou.

- Zde se nenastavuje seznam řetězců, ale menu. Jak vyrobit menu, ukážeme později.

Slide 58

Seznamy a stromy (GtkTreeView)

- Seznam je posloupnost řádků, data v řádcích jsou uspořádána do sloupců (všechny položky ve sloupci mají stejný typ).
- Strom se zobrazuje podobně jako seznam, s odsazením podle hloubky uzlu.
- Rozdělení do 4 částí:
 - Widget (GtkTreeView) ... zobrazuje seznam nebo strom
 - Sloupec (GtkTreeViewColumn) ... objekt obsahující informace o jednom sloupci v seznamu nebo stromu (nadpis sloupce, renderer, specifikace položky datové struktury řádku, který je zobrazen ve sloupci)
 - Renderer (GtkCellRenderer) ... objekt pro kreslení položek
 - Model (GtkListStore, GtkTreeStore) ... datová struktura obsahující celý seznam nebo strom

- Interface pro manipulaci se seznamy a stromy je hodně velký. Více se mu budeme věnovat na cvičení.
- GtkTextView nahradilo widgety GtkList, GtkCList, GtkTree, GtkCTree používané v GTK+ 1.2.

Menu

```
GtkWidget* gtk_menu_bar_new(void);
```

- Vytvoří vodorovný pruh menu, který se obvykle umísťuje k hornímu okraji hlavního okna aplikace.

```
GtkWidget* gtk_menu_item_new_with_mnemonic(const gchar *l);
```

- Vytvoří položku menu.
- Při výběru položky se pošle signál "activate".

```
void gtk_menu_bar_append(GtkMenuBar *menu_bar,  
                          GtkWidget *child);
```

- Vloží položku do menu baru za existující položky.

Slide 59

- Existují i funkce pro vkládání položek do menu před ostatní položky nebo na místo dané indexem položky.
- Třída `GtkMenuItem` je odvozena od `GtkBin`. Položka menu může tedy obsahovat libovolný widget. Obvyčejná položka menu obsahuje statický text (`GtkLabel`). Speciální typy položek menu (odvozené od `GtkMenuItem`) jsou:
 - `GtkCheckMenuItem` ... položka se zaškrťávkem (check boxem)
 - `GtkRadioMenuItem` ... položka s radio buttonem, vždy je aktivní jedna ze skupiny
 - `GtkTearoffMenuItem` ... položka fungující jako odtrhávátko pro odtrhávací menu; jméno odtřené menu se nastavuje pomocí `gtk_menu_set_title()`
- Separátor (vodorovná čára, která nejde vybrat) mezi položkami menu se vyrobí funkcí `gtk_separator_menu_item_new()`.

Menu (2)

```
GtkWidget*gtk_menu_new(void);
```

- Vytvoří submenu (s položkami pod sebou).

```
void gtk_menu_append(GtkMenu *menu, GtkWidget *child);
```

- Přidá položku do menu.

```
void gtk_menu_item_set_submenu(GtkMenuItem *menu_item,  
                               GtkWidget *submenu);
```

- K položce menu připojí submenu, které se automaticky otevře při výběru položky.

Slide 60

- Typická aplikace má hlavní menu typu `GtkMenuBar`. Při výběru položek z hlavního menu se otvírají submenu typu `GtkMenu`. Ta mohou mít připojena submenu další úrovně.

Menu (3)

```
void gtk_menu_popup(GtkMenu *m, GtkWidget *parent_shell,  
    GtkWidget *parent_item, GtkMenuPositionFunc func,  
    gpointer data, guint btn, guint32 activate_time);
```

- Zobrazí menu, po výběru položky menu automaticky zmizí.
- Používá se pro popup menu. Např. následující handler události zobrazí menu na pozici kurzoru myši při stisku třetího tlačítka myši.

Slide 61

```
gint popup(GtkWidget *widget, GdkEventButton *event)  
{  
    if(event->button == 3) {  
        gtk_menu_popup(GTK_MENU(widget), NULL, NULL, NULL,  
            NULL, event->button, event->time);  
        return TRUE;  
    } else return FALSE;  
}
```

- `parent_shell` ... `GtkMenuShell` (`GtkMenu` nebo `GtkMenuBar`), v němž je položka, která vyvolala zobrazení tohoto menu
- `parent_menu_item` ... položka, která vyvolala zobrazení tohoto menu
- `func` ... funkce, která řídí umístění menu, implicitní je na pozici kurzoru myši
- `data` ... argument pro `func`
- `btn` ... tlačítko myši, které iniciovalo zobrazení menu
- `activate_time` ... čas, kdy nastala událost aktivující menu

Akcelerátory

- Klávesové zkratky pro výběr položek menu
- Vytvoření skupiny akcelerátorů pro hlavní okno aplikace:

```
accel = gtk_accel_group_new();  
gtk_window_add_accel_group(GTK_WINDOW(window), accel);
```

- Vytvoření položky menu s akcelerátory:

```
item = gtk_menu_item_new_with_mnemonic("_Open");  
gtk_widget_add_accelerator(item, "activate", accel,  
GDK_o, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
```

- Obvyklý způsob je, že v menu baru mají položky nastavené akcelerátory *Alt+písmeno*, v dalších úrovních jen písmeno. V našem příkladě bude "Open" v submenu "File" a půjde vybrat pomocí sekvence *Alt+F O*. Tyto akcelerátory jsou vyznačeny pomocí podtržení příslušného písmene, což zajistí `gtk_menu_item_new_with_mnemonic()`.
- Druhý akcelerátor umožňuje vybrat položku menu, i když příslušné submenu není zobrazeno. Jeho nastavení se zobrazuje u položky.

Generování menu

- Definice menu

```
static GtkItemFactoryEntry items[] = {
    { "/_File", NULL, NULL, 0, "<Branch>", NULL },
    { "/File/_New", "<control>N", menu_sel, 1, NULL, NULL },
    ... };
```

- Vytvoření skupiny akceleratorů

```
accel = gtk_accel_group_new();
gtk_window_add_accel_group(GTK_WINDOW(window), accel);
```

- Vytvoření menu

```
factory = gtk_item_factory_new(GTK_TYPE_MENU_BAR, "<main>",
    accel);
gtk_item_factory_create_items(factory,
    sizeof(items)/sizeof(*items), items, data);
bar = gtk_item_factory_get_widget(factory, "<main>");
```

Slide 63

- Menu je popsáno polem struktur pro jednotlivé položky:

```
struct GtkItemFactoryEntry {
    gchar *path;
    gchar *accelerator;
    GtkItemFactoryCallback callback;
    guint          callback_action;
    /* possible values:
    * NULL          -> "<Item>"
    * ""           -> "<Item>"
    * "<Title>"    -> create a title item
    * "<Item>"     -> create a simple item
    * "<ImageItem>" -> create an item holding an image
    * "<StockItem>" -> create an item holding a stock image
    * "<CheckItem>" -> create a check item
    * "<ToggleItem>" -> create a toggle item
    * "<RadioItem>" -> create a radio item
    * <path>       -> path of a radio item to link against
    * "<Separator>" -> create a separator
    * "<Tearoff>"  -> create a tearoff separator
    * "<Branch>"   -> create an item to hold sub items
    * "<LastBranch>" -> create a right justified item to hold sub items
    */
    gchar          *item_type;
    /* Extra data for some item types:
    * ImageItem   -> pointer to inlined pixbuf stream
    * StockItem   -> name of stock item
    */
    gconstpointer extra_data;
};
```

Slide 64

Zobrazení a editace textu (GtkTextView)

- Zobrazení a editace textu řídí widget `GtkTextView`.
- Interně používá UTF-8. Glib obsahuje funkce pro překódování mezi aktuálním locale a UTF-8.
- Text je uložen v objektu `GtkTextBuffer`.
- Iterátor (`GtkTextIter`) ... reprezentuje pozici mezi dvěma znaky; zneplatní se při jakémkoliv změně textu.
- Značka (`GtkTextMark`) ... pozice v textu (neviditelný kurzor) platná i při změně textu; značka "insert" je aktuální pozice viditelného kurzoru, "selection_bound" je druhý konec výběru.
- Tag (`GtkTextTag`) ... atributy řídící vzhled úseku textu

- Interface widgetu `GtkTextView` a souvisejících objektů je hodně velký. Více se mu budeme věnovat na cvičení.

Boxy pro tlačítka (GtkHButtonBox, GtkVButtonBox)

- Speciální typ boxů určený pro vkládání tlačítek.
- Zvětší tlačítko a nechává tím větší prostor kolem nápisu na tlačítku.
- Parametry button boxu lze měnit buď individuálně (`gtk_button_box_set_spacing()`, `gtk_button_box_set_layout()`) nebo se dají nastavit standardní hodnoty pro všechny boxy najednou:

```
void gtk_hbutton_box_set_spacing_default(gint spacing);  
void gtk_hbutton_box_set_layout_default(GtkButtonBoxStyle l);
```

- Možné hodnoty pro `GtkButtonBoxStyle`:
 - `GTK_BUTTONBOX_DEFAULT_STYLE` ... použít standardní nastavení
 - `GTK_BUTTONBOX_SPREAD` ... rovnoměrné rozložení
 - `GTK_BUTTONBOX_EDGE` ... rovnoměrné rozložení, ale první a poslední tlačítko budou těsně u okrajů
 - `GTK_BUTTONBOX_START` ... kumulace u levého (resp. horního) okraje
 - `GTK_BUTTONBOX_END` ... kumulace u pravého (resp. dolního) okraje

Slide 66

Zpracování událostí pro widgety bez X oken (GtkEventBox)

- Aby widget mohl dostávat události, musí mít vlastní X okno, protože události jsou generovány X serverem. Ten pracuje pouze s X okny a neví nic o widgetech.
- Když chceme zpracovávat události pro widgety bez X oken (např. `GtkLabel`), můžeme takový widget vložit do event boxu a události zpracovávat v event boxu.
- Další funkcí event boxu je clipping – ořezávání grafiky na rozměr widgetu.

- Při kreslení do X okna se skutečně vykresluje pouze ta část grafiky, která leží uvnitř okna.
- X okno je obvykle stejně velké jako widget.
- Widgety bez vlastního X okna kreslí do okna rodiče a mohou tak kreslit i mimo své hranice.
- Event box má X okno a tím limituje oblast použitelnou vloženým widgetem pro kreslení.

Umístování widgetů (GtkAlignment, GtkFixed)

```
GtkWidget* gtk_alignment_new(gfloat xalign, gfloat yalign,  
                               gfloat xscale, gfloat yscale);
```

- Nastaví relativní pozici a velikost pro synovský widget.
- Potomek GtkBin, může obsahovat pouze jeden synovský widget.

```
void gtk_fixed_put(GtkFixed *fixed, GtkWidget *widget,  
                   gint16 x, gint16 y);
```

- Vloží widget na zadanou pozici.
- Do jednoho GtkFixed lze vložit několik widgetů.

- Pozice v `GtkAlignment` jsou reálná čísla od 0 (úplně vlevo/nahoře) do 1 (úplně vpravo/dole). Hodnota 0.5 znamená umístění doprostřed.
- Velikosti jsou také od 0 (minimální velikost widgetu) do 1 (widget je roztažen přes celý alignment).
- Velikost widgetů se dá řídit také pomocí `GtkSizeGroup`. To není widget, proto může sdružovat widgety na libovolných místech v hierarchii widgetů. Všechny widgety ve stejné skupině mají stejnou šířku, výšku, nebo oba rozměry. Pro každý rozměr se použije maximum z požadavků na velikost všech widgetů ve skupině.

Rámečky (GtkFrame, GtkAspectFrame)

- Rámeček kolem několika widgetů, jako třeba skupiny radio buttons nebo souvisejících check buttons.
- Potomek GtkBin, pokud je potřeba vložit více widgetů (což je obvyklé), je nutné nejprve vložit jiný kontejner (box, tabulku...) a teprve do něj jednotlivé widgety.
- Rámeček má volitelně na horní hraně titulek.
- Vytvoření rámečku s titulkem ... `gtk_frame_new()`
- Pozice titulku ... `gtk_frame_set_label_align()`
- Nastavení reliéfu rámečku ... `gtk_frame_set_shadow_type()`
- `GtkAspectFrame` navíc dovoluje nastavit poměr délek stran rámečku.

- Parametr `yalign` funkce `gtk_frame_set_label_align()` nastavuje vertikální umístění titulku vůči horní hraně rámečku. Default je 0.5.
- Parametr `xalign` určuje zarovnání od 0 (doleva), přes 0.5 (na střed), až po 1 (doprava).
- Když je titulek zadán jako prázdný řetězec "", tak je rámeček přerušen malou mezerou. Nepřerušená linie rámečku se dá dosáhnout zadáním ukazatele `NULL` jako titulku.
- Typy stínování (reliéfu) jsou:
 - `GTK_SHADOW_NONE` ... bez reliéfu (z rámečku se zobrazí jen titulek)
 - `GTK_SHADOW_IN` ... snížený reliéf celého widgetu
 - `GTK_SHADOW_OUT` ... zvýšený reliéf celého widgetu
 - `GTK_SHADOW_ETCHED_IN` ... snížený reliéf orámování
 - `GTK_SHADOW_ETCHED_OUT` ... zvýšený reliéf orámování

Okno s dvěma panely (GtkHPaned, GtkVPaned)

- Rozdělí okno na dvě části, do každé lze vložit jeden widget.
- Uživatel může měnit velikost panelů posunem separátoru mezi nimi.

```
void gtk_paned_pack1(GtkPaned *paned, GtkWidget *child,  
                    gboolean resize, gboolean shrink);
```

```
void gtk_paned_pack2(GtkPaned *paned, GtkWidget *child,  
                    gboolean resize, gboolean shrink);
```

- Vloží widget do levého (`gtk_paned_pack1()`) nebo pravého (`gtk_paned_pack2()`) panelu.

```
void gtk_paned_set_position(GtkPaned *paned, gint position);
```

- Nastavení pozice separátoru mezi panely

Slide 69

- Parametry `gtk_paned_pack1()` a `gtk_paned_pack2()`:
 - `resize ...` zda se má synovský widget zvětšit při zvětšení panelu
 - `shrink ...` zda se synovský widget může zmenšit pod svou minimální velikost

Posuvná oblast (GtkScrolledWindow)

- Přidává scrollbar k synovskému widgetu.
- Některé widgety přímo podporují scrollování, tj. umí se napojit na adjustmety pomocí handleru signálu, jehož id je v `GtkWidgetClass.set_scroll_adjustments_signal`. Takové widgety je možné vložit rovnou do `GtkScrolledWindow`.
- Pro ostatní widgety je nutné mezi widget a `GtkScrolledWindow` vložit `GtkViewport`:

```
void gtk_scrolled_window_add_with_viewport(  
    GtkScrolledWindow *scrolled_window, GtkWidget *child);
```

- Viewport vytvoří velké GDK okno (o velikosti synovského widgetu) a scrollování implementuje posouváním tohoto okna.
- Nevhodné pro widgety, které samy podporují scrollování, např. `GtkTreeView` vložený prostřednictvím viewportu by byl posouván celý včetně nadpisů sloupců. Takové widgety se vkládají rovnou pomocí `gtk_container_add()`.
- Widgety, které podporují scrollování, mají platný identifikátor signálu, tj. nenulovou hodnotu, v položce `GtkWidgetClass.set_scroll_adjustments_signal`.

Slide 71

Toolbar (GtkToolbar)

```
GtkWidget* gtk_toolbar_new(GtkOrientation orientation,  
                           GtkToolbarStyle style);
```

- Vytvoří toolbar.
- *orientation* ... horizontální nebo vertikální
- *style* ... tlačítka na toolbaru budou mít zobrazené buď nápisy, nebo ikony, nebo oboje

```
GtkWidget* gtk_toolbar_append_item(GtkToolbar *toolbar,  
                                   const char *text, const char *tooltip_text,  
                                   const char *tooltip_private_text, GtkWidget *icon,  
                                   GtkSignalFunc callback, gpointer user_data);
```

- Vloží tlačítko do toolbaru, zadává se nápis na tlačítku, ikona, tooltip a callback volaný při stisku tlačítka.

- Při vkládání tlačítek se vždy zadává nápis i ikona, i když se něco z toho nezobrazuje, protože styl zobrazení toolbaru lze měnit voláním `gtk_toolbar_set_style()`.
- Kromě vložení za poslední existující tlačítko funkcí `gtk_toolbar_append_item()` se dá vkládat i na začátek (`gtk_toolbar_prepend_item()`) a na zadanou pozici (`gtk_toolbar_insert_item()`).

Toolbar (2)

```
GtkWidget* gtk_toolbar_append_element(GtkToolbar *toolbar,  
    GtkToolbarChildType type, GtkWidget *widget,  
    const char *text, const char *tooltip_text,  
    const char *tooltip_private_text, GtkWidget *icon,  
    GtkSignalFunc callback, gpointer user_data);
```

- Podle parametru `type` vkládá mezeru, obvyčejné tlačítko, toggle button, radio button, nebo libovolný widget.
- Význam dalších parametrů závisí na hodnotě `type`.

```
void gtk_toolbar_append_space(GtkToolbar *toolbar);
```

- Vloží mezeru, která bude oddělovat následující prvek toolbaru od předchozího.

- I pro widgety a mezery existují funkce pro vkládání na začátek a na zadanou pozici.
- Parametry `gtk_toolbar_append_element()`:
 - `widget` ... používá se při `type==GTK_TOOLBAR_CHILD_WIDGET`, pro ostatní typy se nepoužívá
 - Text, ikona a callback se použijí, pokud `type` není `GTK_TOOLBAR_CHILD_WIDGET` ani `GTK_TOOLBAR_CHILD_SPACE`.

Slide 73

Odtrhávací kontejner (GtkHandleBox)

- Kontejner pro jeden widget (potomek GtkBin)
- Na jedné straně zobrazuje držátko, za které se dá „odtrhnout“ od okna a umístit samostatně kamkoliv na obrazovku.
- Přesunutím na původní místo se „připojí“ zpět do okna.
- Pozice držátka se nastavuje funkcí `gtk_handle_box_set_handle_position()`.
- Odtžený handle box po sobě na původním místě zanechá zbytek nazývaný ghost. S ním musí být ztotožněna jedna hrana boxu, aby byl opět připojen. Která hrana to je, nastaví GTK+ automaticky nebo to lze změnit voláním funkce `gtk_handle_box_set_snap_edge()`.
- Při odtržení handle box emituje signál "child-detached", při připojení "child-attached".

- Vhodný kandidát na umístění do handle boxu je toolbar nebo menu.

Slide 74

Kolekce stránek (GtkNotebook)

- Vždy je vidět jedna stránka, z ostatních jsou vidět záložky s nadpisy.
- Na každé stránce je jeden synovský widget.
- Klepnutím na záložku se zobrazí příslušná stránka.
- Pokud je záložek příliš mnoho, zobrazí se šipky pro scrollování záložek (aktivuje se funkcí `gtk_notebook_set_scrollable()`).
- Stisk pravého tlačítka myši na libovolné záložce zobrazí menu se všemi záložkami.

```
void gtk_notebook_append_page(GtkNotebook *notebook,  
                               GtkWidget *child,  
                               GtkWidget *tab_label);
```

- Přidá další stránku, zadává se synovský widget a nadpis pro záložku.

- Existuje mnoho dalších funkcí pro přidávání a odebrání stránek a nastavování vzhledu notebooku (např. na které straně a zda vůbec se zobrazují záložky).

Dialogové okno (GtkWindow, GtkDialog)

- Funkce void `gtk_dialog_new(void)` vytvoří dialogové okno.
- Třída `GtkDialog` je toplevel okno, které má dvě části oddělené horizontálním separátorem:
 - `hbox (GtkDialog.action_area)` ... pro tlačítka na zavření dialogu ("OK", "Cancel", apod.)
 - `vbox (GtkDialog.vbox)` ... pro ostatní widgety v dialogu
- Zavření dialogu nebo stisk tlačítka generuje signál "response".
- **Nemodální dialog** ... další toplevel okno programu, nebrání interakci uživatele se zbytkem programu.
- **Modální dialog** ... dokud je zobrazen, uživatel nemůže pracovat v ostatních oknech programu.

- Do `action_area` se vkládají tlačítka buď funkcí `gtk_dialog_new_with_buttons()` při vytvoření dialogu, nebo později funkcemi `gtk_dialog_add_button()` (vloží jedno tlačítko, vrací ukazatel na něj), nebo `gtk_dialog_add_buttons()` (vloží několik tlačítek, nevrací nic).
- Pokud to jde, je vhodné preferovat nemodální dialogy, protože neomezují uživatele.
- Modální dialog je vhodné použít pouze tehdy, pokud je potřeba, aby uživatel zareagoval na dialog dřív, než udělá s programem jakoukoliv další akci.

Modální dialog

```
resp = gtk_dialog_run(GTK_DIALOG(dialog));
switch(resp) {
    case GTK_RESPONSE_NONE: ...
    case GTK_RESPONSE_YES : ...
    case GTK_RESPONSE_NO  : ...
    case GTK_RESPONSE_DELETE_EVENT: ...
    default:
}
gtk_widget_destroy(dialog);
```

Slide 76

- Funkce `gtk_dialog_run()` nastaví dialog jako modální, zobrazí ho a spustí vnořenou `gtk_main()`. Po stisku tlačítka vrátí příslušný `GTK_RESPONSE_*` kód přiřazený tlačítku při jeho vložení do dialogu. Návrátová hodnota `GTK_RESPONSE_DELETE_EVENT` znamená zavření dialogu window managerem, `GTK_RESPONSE_NONE` je zrušení dialogu jiným způsobem.

Další widgety

- pravítka (GtkHRuler, GtkVRuler)
- kalendář (GtkCalendar)
- dialog pro zobrazování různých hlášení programu (GtkMessageDialog)
- dialog pro výběr souboru (GtkFileSelection)
- dialog pro nastavení barev (GtkColorSelectionDialog)
- dialog pro výběr fontu (GtkFontSelectionDialog)
- dialog pro nastavení XInput extension (GtkInputDialog)

- *Pravítka* ukazují pozici myši v okně, používají se např. v kreslicích programech (Gimp).
- *Kalendář* zobrazuje jeden měsíc z kalendáře, umožňuje pohyb po měsících a letech a vybírání (označování) jednotlivých dnů.
- *Message dialog* se používá na zobrazování hlášek a dotazů programu.
- *Dialog pro výběr souborů* je předpřipravený dialog pro zadání jména souboru včetně seznamu s obsahem aktuálního adresáře, tlačítka „OK“, „Cancel“, „Help“ a tlačítka pro smazání a přejmenování souboru a pro vytvoření adresáře.
- *Dialog pro nastavení barev* je standardní dialog pro výběr barvy v prostoru RGB nebo HSV včetně nastavení průhlednosti.
- *Dialog pro výběr fontu* je standardní dialog zadání textového fontu s možností zadat filtr pro seznam nabízených fontů.
- *Dialog pro nastavení XInput extension* je předpřipravený dialog pro nastavování parametrů vstupních zařízení.

Slide 78

GLib

- Knihovna pomocných funkcí a datových struktur
- Přenositelná, funguje na unixových systémech a na Windows.
- Poskytuje náhradu pro některé standardní typy a pro některé funkce ze standardní knihovny jazyka C.
- Typy: `gint8`, `guint8`, ..., `gint64`, `guint64`; `gboolean`, `gchar`, `gint`, `gpointer`, ...
- Převody integer ↔ pointer: makra `GINT_TO_POINTER(i)`, `GPOINTER_TO_INT(p)`, `GUINT_TO_POINTER(i)`, `GPOINTER_TO_UINT(p)`
- Ladicí makra `g_return_if_fail(cond)` a `g_return_val_if_fail(cond, retval)`

- Některé funkce (např. `g_strcasecmp()`) jsou jen jinak pojmenované standardní funkce. V GLib jsou kvůli přenositelnosti programů na platformy, které standardní varianty těchto funkcí nepodporují.
- Pro `gboolean` jsou definovány konstanty `FALSE` a `TRUE`.
- Typy s udanou šířkou v bitech odpovídají standardním typům z `inttypes.h`: `int8_t`, `uint8_t`, ..., `int64_t`, `uint64_t`.
- Makra `g_return_if_fail` a `g_return_val_if_fail` se používají pro kontrolu parametrů ve funkcích.
- Další ladicí makra: `g_assert`, `g_assert_not_reached`
- Makra `MAX(a, b)`, `MIN(a, b)`, `ABS(x)` a `CLAMP(x, low, high)`

Alokace paměti

```
gpointer g_malloc(gulong size);
```

- Alokuje paměť.
- Pokud nelze alokovat, ukončí program.
- Pro `size == 0` vrací NULL.

```
gpointer g_malloc0(gulong size);
```

- Alokuje paměť a vynuluje ji.

```
void g_free(gpointer mem);
```

- Dealokuje paměť.
- Pro `mem == NULL` nedělá nic.

Slide 79

- Alokační a dealokační funkce podporují ladění a profilování práce s pamětí.
- Je třeba správně párovat funkce: `malloc()` ↔ `free()`, `g_malloc()` ↔ `g_free()`, `new` ↔ `delete`, protože každá dvojice může používat jiný memory pool.

Řetězce

- Varianty obvyklých funkcí pro práci s řetězcí, používají typ `gchar *`: `g_snprintf()`, `g_strcasecmp()`, `g_strdup()`, atd.
- Datový typ **GString** – buffer, který se automaticky zvětšuje při přidávání textu
- Místo alokované pro text v GString roste po mocninách dvou.
- Funkce pro manipulaci s GString mají prefix `g_string_`.
- Příklady funkcí: `g_string_new()`, `g_string_sprintf()`, `g_string_insert()`, `g_string_erase()`

Slide 80

- Typ GString má public reprezentaci

```
struct _GString
{
    gchar *str;
    gsize len;
    gsize allocated_len;
};
```

obsahující ukazatel na data, délku řetězce (bez ukončující nuly) a počet bajtů alokovaných pro `str`.

Seznamy

- Prázdný seznam reprezentovaný ukazatelem NULL.
- Funkce pro přidávání a odebrání prvků, iterované volání funkce pro všechny prvky, atd.
- Jednosměrný seznam GSList:

```
GList* list = NULL;  
gchar* element1 = g_strdup("prvni");  
gchar* element2 = g_strdup("druhy");  
list = g_slist_append(list, element1);  
list = g_slist_prepend(list, element2);
```

- Obousměrný seznam (GList) je podobný, navíc má jen makro `g_list_previous(list)` pro přístup k předchozímu prvku.

Slide 81

- GLib udržuje pouze ukazatel na začátek seznamu, takže operace `g_list_append()` pro seznam délky n má složitost $O(n)$.
- Je lepší seznam stavět od konce pomocí funkce `g_list_prepend()`, která funguje v čase $O(1)$ a pak ho případně obrátit voláním `g_list_reverse()`.

Slide 82

Další datové struktury

- **GTree** ... vyvážený binární strom
- **GNode** ... obecný n -ární strom
- **GHashTable** ... hašovací tabulka
- **GArray** ... pole hodnot libovolného typu, které automaticky roste při přidávání prvků
- **GQuark** ... asociace řetězce a číselného identifikátoru
- **datasets** ... asociace datových struktur s nějakou adresou v paměti
- **GCache** ... sdílení datových struktur

- Quark je obdoba X-ového atomu.
- Dataset se použije, když potřebujeme k nějaké struktuře, kterou nemůžeme modifikovat, připojit další data. Adresa struktury se použije jako klíč pro vyhledání přidaných dat v datasetu.
- GTK+ používá GCache pro sdílení stylů a grafických kontextů.

Slide 83

Další funkce

- **GHookList** ... generická podpora callback funkcí
- dynamické načítání modulů (plug-ins)
- generování logovacích hlášení
- časovače
- generický cyklus zpracování událostí
- lexikální analyzátor
- automatické doplňování řetězců

- Smyčka událostí v GLib slouží jako základ pro `gtk_main()`.
- Lexikální analyzátor je pevně definovaný, není to tedy náhrada např. za `flex`.
- Lexikální analyzátor se používá při čtení GTK+ resource souborů.
- Doplňování řetězců se typicky používá pro doplňování jmen souborů, např. v dialogu `GtkFileSelection`.

Slide 84

GDK

- Většina funkcí GDK jsou wrappery funkcí z Xlib.
- Skrývá některé vlastnosti Xlib, tím usnadňuje používání GDK a její portování na jiné okenní systémy (např. Windows).
- Každá datová struktura má veřejnou verzi (definovanou v `gdk.h`) a privátní verzi, která navíc obsahuje položky specifické pro okenní systém, nad kterým GDK funguje. Například funkce `gdk_window_new()` pro vytvoření GDK okna vrací ukazatel na `GdkWindow` a odpovídající privátní struktura `GdkDrawableImplX11` obsahuje atributy příslušného X okna.
- Při programování nových tříd widgetů jsou z GDK nejvíce potřeba funkce pro vytváření oken a kreslení.

- GDK se nebudeme dále zabývat. Pro hlubší pochopení GDK je třeba vědět, jak funguje Xlib a celý X Window System. Až probereme Xlib, bude i fungování GDK celkem zřejmé, protože převážně pouze obaluje funkce Xlib.
- Některé ukázky použití GDK uvidíme, až budeme popisovat fungování widgetů.

Timeouts

```
guint gtk_timeout_add(guint32 interval, GtkFunction func,  
                       gpointer data);
```

- Zaregistruje funkci `func()`, která bude opakovaně volána s parametrem `data` s periodou `interval` milisekund.
- Vrací id registrace.

```
void gtk_timeout_remove(guint timeout_handler_id);
```

- Zruší registraci funkce s daným id. Funkce už nebude volána.
- Odregistrovat funkci pro timeout lze i tím, že funkce vrátí `FALSE`. Pokud funkce vrátí `TRUE`, bude se dál volat po uplynutí nastavené periody.

Slide 85

- Pomocí timeoutů můžeme ošetřit periodické akce, např. nějakou animaci nebo posun progress baru.

Idle functions

```
guint gtk_idle_add(GtkFunction function, gpointer data);
```

- Zaregistruje funkci, která se bude volat, když na zpracování nečekají žádné události s vyšší prioritou.
- Vrací registrační id.

```
void gtk_idle_remove(guint idle_handler_id);
```

- Odregistruje funkci.
- Funkce je také odregistrována, pokud vrátí FALSE.

Slide 86

- Jednotlivé operace prováděné v rámci `gtk_main()` mají různé priority.
- Standardní priorita pro idle funkce je nízká.

I/O kanály

```
GIOChannel* g_io_channel_unix_new(int fd);
```

- Vytvoří I/O kanál pro zadaný deskriptor souboru.
- Kódování dat v kanálu se nastavuje funkcí `g_io_channel_set_encoding()`. Default je UTF-8. Pro binární data je třeba nastavit NULL.

Slide 87

```
guint g_io_add_watch(GIOChannel *chan, GIOCondition cond,  
                    GIOFunc func, gpointer user_data);
```

- Zaregistruje kanál do hlavní smyčky událostí. Pokud na kanálu nastane zadaná podmínka, zavolá se funkce `func`.

```
void g_io_channel_unref(GIOChannel *channel);
```

- Sníží počet odkazů na kanál. Při zrušení posledního odkazu se kanál smaže.

- Podmínky, na které lze čekat, jsou:
 - `G_IO_IN` ... možnost číst
 - `G_IO_OUT` ... možnost zapisovat
 - `G_IO_PRI` ... možnost číst urgentní data
 - `G_IO_ERR` ... nastala chyba
 - `G_IO_HUP` ... ukončení spojení
 - `G_IO_INVALID` ... chyba, deskriptor není otevřený

- Funkce volaná při splnění podmínky je typu

```
gboolean (*GIOFunc)(GIOChannel *src,  
                   GIOCondition cond, gpointer data);
```

- Tyto funkce se hodí např. pokud program komunikuje po síti. Program by se neměl zablokovat čekáním na příchod nebo odeslání dat, protože mezitím by nefungovalo uživatelské rozhraní.
- Využívá se toho, že X používá síťový protokol. Události přicházejí od X serveru přes socket. GLib volá při čekání na událost (od X serveru a na zaregistrovaných kanálech) funkci `poll()` a podle toho, jestli přijde událost X nebo nastane aktivita na jiném deskriptoru, zavolá buď zpracování události nebo registrovanou funkci pro I/O kanál.

Slide 88

Resource files

- Textové soubory, kterými lze ovlivňovat vzhled (styl) programu.
- Základní vzhled widgetů je definován přímo v knihovně GTK+ nebo pomocí theme.
- Styl definuje barvy, fonty, pixmapy na pozadí a přiřazení akčních signálů klávesám.
- Resource soubory se načítají funkcí `gtk_rc_parse()`, která se obvykle volá po `gtk_init()`. Navíc se na konci `gtk_init()` načítají soubory `<SYSCONFDIR>/gtk-2.0/gtkrc` a `~/.gtkrc-2.0`, kde `<SYSCONFDIR>` je standardně `/usr/local/etc`.
- Navíc se ještě čte soubor specifický pro nastavené locale, např. `~/.gtkrc-2.0.cs_CZ`.

- Resource soubory v GTK+ mají pevně dané typy hodnot, které mohou obsahovat. Nedají se tedy přímo použít jako obecné konfigurační soubory pro aplikace tak, jako lze používat X resource files.
- Akční signály jsou signály, které lze namapovat na klávesy, protože před jejich vyvoláním není třeba provádět žádné speciální přípravy. Stiskem klávesy se vygeneruje připojený signál. V resource souboru je vždy klávesa, jméno signálu a případně parametry signálu.

Slide 89

Obsah RC souboru

- Definice stylu

```
style "my-menu"  
{  
  font="-*-arial-medium-r-*-*-*120-*-*p*-iso8859-2"  
  bg[PRELIGHT] = { 0.0, 0.0, 0.6 }  
  fg[PRELIGHT] = { 1.0, 1.0, 1.0 }  
}
```

- Barvy se definují pro jednotlivé stavy widgetů:

- NORMAL ... normální stav
- ACTIVE ... aktivovaný widget, např. kliknutím myši
- PRELIGHT ... tlačítko nebo prvek menu, na kterém je kurzor myši
- SELECTED ... vybrané položky v seznamu, označený text v editačním řádku, apod.
- INSENSITIVE ... widgety s vypnutou reakcí na akce uživatele

- Takto se definuje pojmenovaný styl, který se dále přiřadí určitým widgetům.
- Kromě font ještě existují fontset a font_name (Pango font name). Když se vyskytne více definic fontu, font má nejmenší a font_name největší prioritu.

Obsah RC souboru (2)

- Přiřazení stylu widgetu podle jeho jména a jmen nadřazených widgetů. Funkce `gtk_widget_set_name()` nastaví jméno widgetu.
`widget "mywindow.*.GtkMenuItem" style "my-menu"`
- Přiřazení stylu widgetu podle jeho třídy a tříd nadřazených widgetů
`widget_class "GtkWindow.*.GtkMenuItem" style "my-menu"`
- Přiřazení stylu všem widgetům dané třídy a odvozených tříd
`class "GtkMenuItem" style "my-menu"`

- Cesta od toplevel okna k widgetu je pro přiřazení pomocí `widget` tvořena posloupností jmen widgetů (pokud některý widget nemá nastavené jméno, použije se jméno třídy) oddělených tečkami. V RC souboru se v cestě dají používat wildcardy "*" (zastupuje lib. posloupnost znaků) a "?" (zastupuje jeden znak).
- Cesta pro `widget_class` je obdobná, ale vždy tvořena jmény tříd i pro widgety, které mají definované jméno widgetu.
- Všechny položky všech načtených RC souborů, které odpovídají určitému widgetu, se skládají tak, že nejvyšší prioritu mají položky `widget`, následuje `widget_class` a nakonec `class`. V rámci jednoho typu položek mají přednost ty, které se načetly později.
- Mapování kláves funguje podobně: přiřazení kláves a signálů definovaná položkami `binding` se pomocí `widget`, `widget_class` a `class` přiřazují widgetům. Navíc se dá definovat priorita, se kterou se mapování uplatní. Standardně mají nejnižší prioritu klávesy definované interně v GTK+, následují klávesy definované v programu a nejvyšší prioritu mají klávesy z RC souborů.

Slide 91

Komunikace mezi programy

- **Selections** . . . Uživatel v okně jedné aplikace vybere nějaká data (např. označí blok textu) a kliknutím do editačního widgetu v jiné aplikaci je vloží.
- **Drag&drop** . . . Uživatel přetáhne myší ikonu reprezentující vybraný objekt z jednoho okna do jiného, což způsobí přenos dat.
- Data lze přenášet v různých formátech. Zdroj nabízí nějaké formáty a cíl si vybere, v jakém formátu data chce.
- Programy komunikují prostřednictvím X serveru ⇒ mohou běžet na různých počítačích, nenavazuje se mezi nimi přímé spojení.
- Používají se standardní komunikační mechanismy X ⇒ GTK+ aplikace si může vyměňovat data i s programy používajícími jiné toolkity nebo dokonce založenými přímo na Xlib.

- Widgety jako `GtkEntry` mají zabudovanou podporu selections.
- Mechanismus selections lze využít i pro komunikaci plně řízenou zúčastněnými programy, bez toho, aby uživatel musel něco označovat a bez jakékoliv vizuální indikace (obarvení označeného bloku) přenášených dat.
- V GTK+ 2 je navíc clipboard (`GtkClipboard`), který je implementovaný pomocí selections.

Atom

- Číselný identifikátor přiřazený nějakému řetězci
- Tabulka přiřazení mezi řetězci a atomy je uložena v X serveru ⇒ všechny programy pracující se stejným X serverem dostanou pro určitý řetězec stejný atom.
- Různé řetězce mají různé atomy.

Slide 92

```
GdkAtom gdk_atom_intern(const gchar *atom_name,  
                        gint only_if_exists);
```

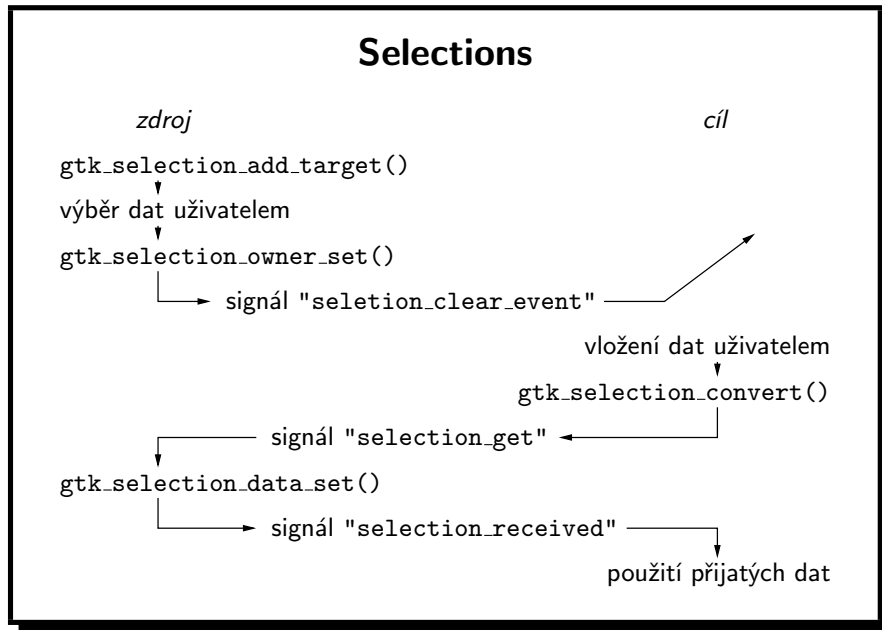
- Vrátí atom pro řetězec atom_name.
- Jestliže atom neexistuje a only_if_exists je FALSE, vytvoří se nový atom.

```
gchar* gdk_atom_name(GdkAtom atom);
```

- Vrátí jméno atomu.

- Atomy dovolují místo dlouhých řetězců používat typicky mnohem kratší čísla.
- Atomy se obvykle používají jako identifikátory sdílených systémových prostředků.
- Získávání atomů pomocí řetězců snižuje nebezpečí výskytu konfliktů, jaké se často vyskytují např. mezi klíči pro prostředky System V IPC.

Slide 93



- Tento obrázek ukazuje posloupnost akcí na straně vlastníka a příjemce výběru při předávání dat z výběru.

Selections (2)

```
gint gtk_selection_convert(GtkWidget *widget,  
    GdkAtom selection, GdkAtom target, guint32 time);
```

- Požádá o vybraná data.
- Obvykle se používá výběr (*selection*) GDK_PRIMARY_SELECTION.
- *target* ... požadovaný typ dat, např. "STRING" nebo "INTEGER"
- *time* ... čas události, která vyvolala žádost o data, nebo konstanta GDK_CURRENT_TIME
- Dodání dat vlastníkem výběru je oznámeno signálem "selection_received". Hlavička příslušného handleru je:

```
void selection_received(GtkWidget *widget,  
    GtkSelectionData *sel_data, gpointer data);
```

Slide 94

- Kromě GDK_PRIMARY_SELECTION lze použít i libovolné jiné výběry.
- Cíl "TARGETS" vrátí seznam možných typů (ve formě pole atomů), ve kterých je vlastník výběru schopen poskytnout data.
- Ve struktuře GtkSelectionData jsou zajímavé položky:
 - *type* ... atom identifikující typ přijatých dat
 - *data* ... ukazatel na data
 - *length* ... délka dat v bajtech nebo záporné číslo při chybě

Selections (3)

```
void gtk_selection_add_target(GtkWidget *widget,  
    GdkAtom selection, GdkAtom target, guint info);
```

- Reguluje cíl (typ dat) poskytovaný widgetem pro určitý výběr.
- Nutno volat pro každou dvojici (výběr, cíl) poskytovanou widgetem.

```
gint gtk_selection_owner_set(GtkWidget *widget,  
    GdkAtom selection, guint32 time);
```

- Přivlastnění výběru
- Předchozí vlastník dostane "selection_clear_event".

Slide 95

- Seznam registrovaných cílů widget automaticky vrací jako odpověď na žádost o cíl "TARGETS".
- Když v `gtk_selection_owner_set()` je `widget == NULL`, program se vzdá vlastnictví výběru.
- Program si typicky přivlastní výběr, když uživatel označí nějaká data, např. označí myší úsek textu v terminálovém okně.

Selections (4)

- Když nějaký program zavolá `gtk_selection_convert()`, vlastník výběru dostane signál "selection_get". Prototyp handleru je:

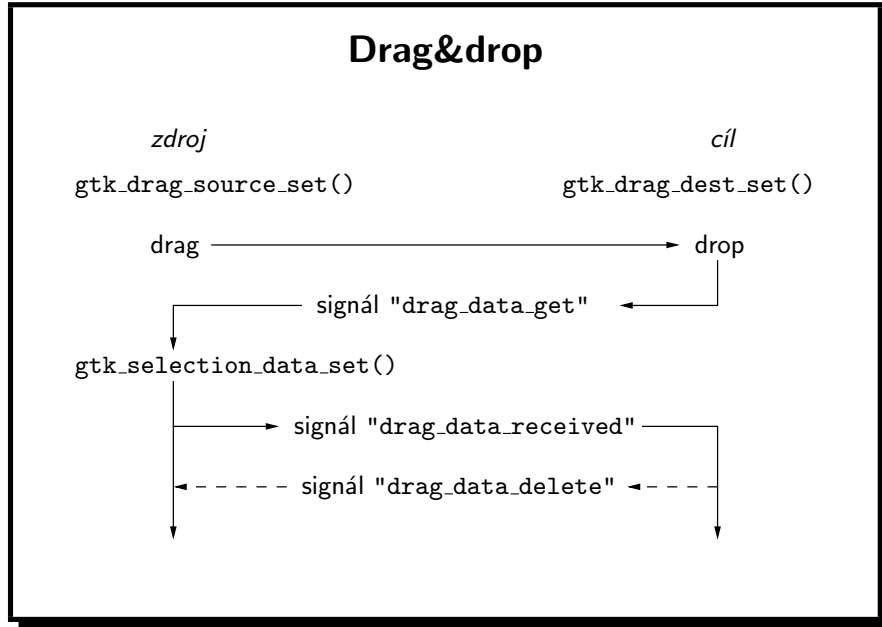
```
void selection_get(GtkWidget *widget,  
                  GtkSelectionData *sel_data, guint info, guint time);
```
- Handler by měl naplnit ve struktuře `sel_data` položky `type` (poskytnutý cíl – typ dat), `format` (délka jednoho datového prvku v bitech), `data` a `length`. Naplnění struktury provede funkce

```
void gtk_selection_data_set(GtkSelectionData *sel_data,  
                             GdkAtom type, gint format, guchar *data,  
                             gint length);
```
- Data se následně automaticky odešlou příjemci.

Slide 96

- Položka `sel_data->target` je požadovaný cíl (typ dat).
- Parametr `info` v handleru je hodnota `info` nastavená funkcí `gtk_selection_add_target()`.
- Položka `format` ve struktuře `GtkSelectionData` má obvykle hodnotu 8 (znak) nebo 32 (integer). X server ji používá pro úpravu pořadí bitů (big/little endian).

Slide 97



- Funkce `gtk_drag_source_set()` a `gtk_drag_dest_set()` se volají pro widget, který může být zdrojem, resp. cílem operace drag&drop. Definují typy dat (cíle), které widget umí poskytnout/přijmout a typy drag&drop operací (kopie nebo přesun).
- V okamžiku „drop“ (uživatel pustí tažený objekt) dostane zdroj signál "drag_data_get", na který odpoví stejně jako při žádosti o výběr, tj. funkcí `gtk_selection_data_set()` poskytne data. Handler signálu má hlavičku

```
void drag_fun(GtkWidget *widget, GdkDragContext *context,
              GtkSelectionData *selection_data, guint info, guint time,
              gpointer data);
```

- Cíl operace drag&drop následně dostane signál "drag_data_received" – informaci o přijetí dat. Handler tohoto signálu má prototyp

```
void drop_fun(GtkWidget *widget, GdkDragContext *context, gint x,
              gint y, GtkSelectionData *data, guint info, guint time);
```

- Pokud operace byla přesun (`GDK_ACTION_MOVE`), dostane ještě zdroj pomocí signálu `drag_data_delete` informaci, že může smazat přesunutá data. Na signál lze reagovat handlerem s prototypem

```
void del_fun(GtkWidget *widget, GdkDragContext *context, gpointer data);
```

Třídý a objekty

- 2 struktury pro každou třídu odvozenou z GObject:
 - struktura pro **instanci** (např. `GtkButton`) – obsahuje data konkrétní instance
 - struktura pro **třídu** (např. `GtkButtonClass`) – obsahuje ukazatele na funkce, obdoba tabulky virtuálních metod v C++
- První prvek musí být vždy rodičovská struktura, aby šlo přetypovat na rodiče.

Slide 98

```
typedef struct _GtkButton GtkButton;  
typedef struct _GtkButtonClass GtkButtonClass;  
struct _GtkButton          struct _GtkButtonClass  
{                          {  
    GtkBin bin;  
    ...  
};                          };  
                             GtkBinClass parent_class;  
                             ...
```

- Každý typ widgetu má svůj `.h` a `.c` soubor.
- Ve struktuře pro třídu mohou být i data třídy (obdoba `static members` v C++), ale častěji se definují jako `static` proměnné v `.c` souboru pro widget.

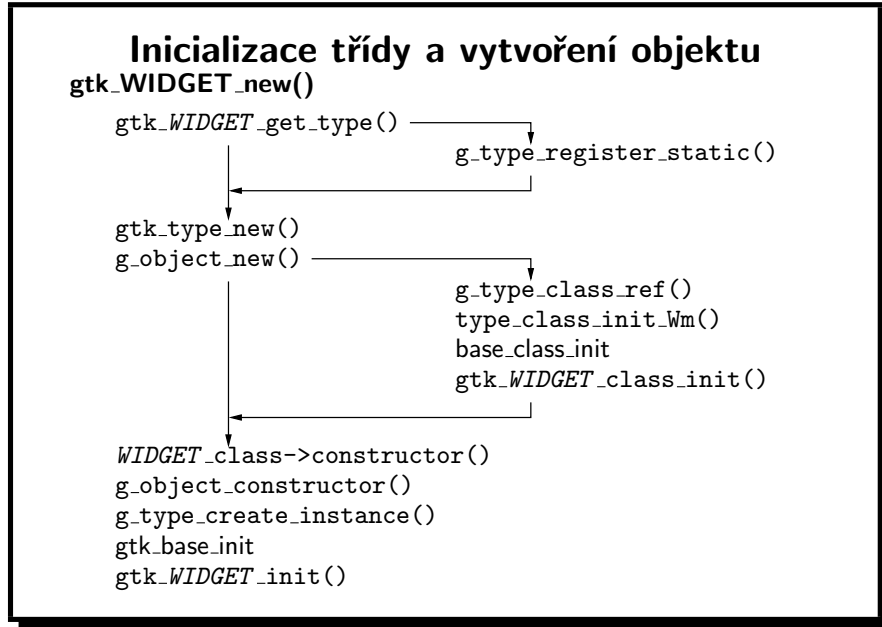
Slide 99

Typová kontrola

- Každý datový typ (nejen objekty) má v GTK+ identifikátor typu (`GtkType`).
- Typ widgetu `widget` vrací funkce `gtk_widget_get_type()`.
- Pro každý typ existují makra (na místě `WIDGET` se dosadí jméno konkrétního typu, např. `BUTTON`):
 - `GTK_TYPE_WIDGET` – vrací id typu
 - `GTK_WIDGET(obj)` – přetypování instance
 - `GTK_WIDGET_CLASS(klass)` – přetypování třídy
 - `GTK_IS_WIDGET(obj)` – test na typ
 - `GTK_IS_WIDGET_CLASS(obj)` – test na třídu

- K identifikátoru typu jsou přiřazeny informace nutné pro inicializaci třídy a vytváření instancí.
- Kvůli kolizi s klíčovým slovem `class` v C++ se používá identifikátor `klass`.
- Typ dostane identifikátor při registraci.
- Každá třída widgetů je automaticky registrována při vytvoření prvního objektu této třídy.
- Fundamentální typy, např. primitivní typy v C (`G_TYPE_INT`) nebo `G_TYPE_OBJECT`, jsou automaticky registrovány ve funkci `gtk_init()`.

Slide 100



- Vytvoření objektu typu *WIDGET* se provede metodou `gtk_WIDGET_new()` příslušné třídy.
- Postup vytvoření objektu je následující:
 1. Zavolá se metoda `gtk_WIDGET_get_type()`, která vrátí identifikátor typu. Pokud se volá poprvé, musí zajistit registraci typu. Ta se provede zavoláním funkce `g_type_register_static()`, která dostane rodičovský typ a strukturu obsahující jméno nového typu, velikosti struktur pro třídu a instance a ukazatele na inicializační funkce.
 2. Identifikátor typu je předán funkci `gtk_type_new()`, která volá `g_object_new()`. Ta před vytvořením první instance inicializuje strukturu pro třídu voláním funkce `g_type_class_ref()`, která dále volá `type_class_init_Wm()`. Tato funkce alokuje paměť pro strukturu třídy a zkopíruje do ní strukturu rodičovské třídy. Pokud nestačí bázovou třídu zkopírovat, lze pro třídu definovat funkci na další inicializaci kopie bázové třídy (tyto funkce se volají postupně v celé posloupnosti bázových tříd ve směru od kořene hierarchie). Nakonec se provede inicializační funkce třídy (zajistí registraci properties a signálů).
 3. Funkce `gtk_type_new()` zavolá konstruktor pro třídu. Konstruktor odvozené třídy nejprve volá konstruktor bázové třídy. Konstruktor `g_object_constructor()` alokuje paměť pro instanci. Pak zavolá postupně inicializační funkce pro objekty bázových tříd a nakonec pro vytvářený objekt.

Slide 101

Properties

- **property** – dvojice klíč (string) a hodnota (GValue)
- Jméno property: *třída::property*, např.
`GtkContainer::border_width`
- Jméno třídy lze vynechat, pak hledá první výskyt property ve směru od třídy objektu ke GObject.
- `g_object_class_list_properties()` – vrací pole všech properties widgetu.
- `g_object_set()` – nastaví hodnoty properties zadaných jako seznam dvojic jméno, hodnota ukončený NULL.
- `g_object_get()` – dostane seznam dvojic jméno, adresa pro uložení hodnoty, vrátí hodnoty properties.

- Funkce `g_object_list_properties()` se dá použít na zjištění, jaké všechny argumenty widget má. Vrací pole struktur `GParamSpec`.

Definice properties

- Properties se definují v inicializační funkci třídy voláním `g_object_class_install_property()`.
- Je třeba definovat funkce `g_WIDGET_get_property()` a `gtk_WIDGET_set_property()`, které obsluhují properties definované v této třídě. Ukazatele na tyto funkce se přiřadí do `((GObject *)WIDGET_klass)->get_property` a `((GObject *)WIDGET_klass)->set_property`.
- Není třeba se starat o zděděné properties, automaticky se pro ně volají správné funkce rodičovských tříd.

- Funkce pro nastavení/přečtení property může ve skutečnosti dělat i něco složitějšího než jen změnu nebo přečtení hodnoty proměnné.
- Navíc se často definují speciální funkce, které fungují stejně jako manipulace s některými properties. Např. `gtk_container_set_border_width()` dělá totéž jako nastavení hodnoty property `GtkContainer::border-width`.

Definice signálů

```
guint g_signal_new(const gchar *name, GType itype,
    GSignalFlags flags, guint off,
    GSignalAccumulator accumulator, gpointer accu_data,
    GSignalCMarshaller mar, GType rt, guint npar, ...);
```

- Registruje nový signál, vrátí identifikátor signálu. Používá se v inicializační funkci třídy.
- name – jméno signálu
- itype – typ, k němuž signál patří
- flags – pořadí spouštění implicitních a uživatelských handlerů
- off – pozice ukazatele na implicitní handler ve struktuře třídy
- accumulator – akumulátor návratových hodnot handlerů
- mar – převodní funkce: pole GValue → seznam parametrů funkce
- rt – typ návratové hodnoty nebo G_TYPE_NONE
- npar – počet parametrů handleru
- ... – seznam typů parametrů (G_TYPE_*)

- Signál může mít parametry, které se handlerům předávají mezi ukazatelem na objekt a generickým ukazatelem zadávaným v `g_signal_connect()`. Parametry a místo pro uložení návratové hodnoty se zadávají při emisi signálu (např. `g_signal_emit()`).
- Jméno signálu se používá v `g_signal_emit_by_name()`.
- Handlers mohou být buď volány rekurzivně, nebo opakovaná emise stejného signálu způsobí restart zpracování signálu.
- Offset handleru se používá pro generické volání handleru.
- V C není možné dynamicky konstruovat seznam parametrů funkce, proto se pro převod z pole argumentů typu `GValue` na seznam parametrů handleru používají *marshaller*. Pro každou používanou kombinaci typů parametrů a návratové hodnoty musí být definován jeden marshaller. V GTK+ jsou definovány nejčastěji využívané marshallery. Pro signály s neobvyklými parametry je třeba definovat vlastní marshallery.
Konvence pro jména marshallerů je `gtk_marshal_RETVAL_ARG1_ARG2..._ARGN`
- Typy parametrů a návratové hodnoty se zadávají pomocí identifikátorů typů.

Slide 104

Zrušení objektu

- Začíná voláním `g_object_unref()` nebo `gtk_widget_destroy()`.
- 1. **Dispose** – akce prováděné před zahájením destrukce objektu; obvykle se nepředefinovává; implicitní metoda emituje signál "destroy" a tím zahájí druhou fázi rušení objektu
- 2. **Destroy** – označí objekt jako „nepoužitelný“ a uvolní s ním asociované zdroje; objekt po destrukci by měl zůstat bezpečný, tj. všechny veřejné metody by měly fungovat i po destrukci
- 3. **Finalize** – volá se, pouze když počet odkazů na objekt dosáhne 0; uvolní strukturu instance objektu.
- První dvě fáze mohou proběhnout i pro objekty s počtem odkazů větším než 0.

- Při předefinování některé z úklidových metod je třeba volat původní metodu rodičovské třídy.
- Jednotlivým fázím rušení objektu odpovídají ukazatele na funkce ve strukturách tříd:

```
void (*dispose)(GObject *object); /* v GObjectClass */  
void (*destroy)(GtkObject *object); /* v GtkObjectClass */  
void (*finalize)(GObject *object); /* v GObjectClass */
```
- Pouze *destroy* je handler (signálu "destroy"), ostatní jsou obyčejné metody.
- Metoda *dispose* v `GtkWidget` vyjme widget z rodičovského kontejneru a zruší jeho X okno (provede `unrealize`).
- Metoda *destroy* kontejnerů zruší všechny synovské widgety.

Slide 105

Připojení dat k objektu

```
void g_object_set_data_full(GObject *object,  
    const gchar *key, gpointer data,  
    GDestroyNotify destroy);
```

- K objektu `object` připojí data identifikovaná klíčem `key`.
- Při nastavení dat na NULL voláním `g_object_set_data()` nebo při zrušení objektu se volá `destroy(data)`.

```
gpointer gtk_object_get_data(GtkObject *object,  
    const gchar *key);
```

- Vrábí data asociovaná s klíčem `key` v objektu `object`.

- Data k objektům je možné přidávat libovolně, není třeba předem deklarovat, jaké klíče se budou používat pro kterou třídu.

Realizace a mapování

- Widget, který má GDK okno a je přímým potomkem GtkWidget, musí definovat metodu `realize()` (implicitní funguje pro widgety bez GDK oken).
- Metoda `realize()` nastaví příznak widgetu `GTK_REALIZED`, vytvoří GDK okno voláním `gdk_window_new()`, ukazatel na něj uloží do `widget->window`, k oknu přidá odkaz na widget pomocí `gdk_window_set_user_data(widget->window, widget)`; a nastaví styl widgetu funkcí `gtk_style_attach()`.
- Metoda `unrealize()` třídy GtkWidget odmapuje GDK okno a zruší ho. Pokud je widget kontejner, tak nejdřív zavolá `unrealize()` pro všechny synovské widgety.
- Metody `map()` a `unmap()` nastaví/zruší příznak `GTK_WIDGET_MAPPED` a zobrazí/schová `widget->window`.

- Styl obsahuje X-ové zdroje, proto se musí vytvořit až po vytvoření X okna.
- Metody `unrealize()`, `map()` a `unmap()` není obvykle nutné předefinovat.

Nastavení velikosti

```
void (*size_request)(GtkWidget *widget,  
                    GtkRequisition *requisition);
```

- Tato metoda by do requisition měla uložit požadovanou velikost widgetu.

```
void (*size_allocate)(GtkWidget *widget,  
                    GtkAllocation *allocation);
```

- Dostane přidělenou velikost v allocation.
- Přiřadí velikost do widget->allocation.
- Přidělí velikosti případným synovským widgetům.
- Upraví velikosti GDK oken, pokud je widget realizován.

Slide 107

- Požadavky na velikost widgetů se propagují směrem k rodičům.
- Top-level okno pak dostane přidělenou velikost.
- Velikosti se rozdělují mezi synovské widgety a propagují se směrem k listům stromu widgetů.

Kreslení

Situace, kdy se překresluje celý nebo část widgetu (pomocí metody `expose_event()` widgetu):

1. Příchod události **expose_event**, jestliže se část GDK okna widgetu stala viditelnou a potřebuje překreslit.
2. GTK+ rozhodne o překreslení: při změně velikosti widgetu, při nahrání nové theme.
3. Widget se sám rozhodne překreslit; lze ošetřit libovolným způsobem.

Slide 108

- Kreslí se pomocí kreslicích funkcí GDK.
- Ještě lepší je, pokud to jde, použít funkce z theme, aby vzhled widgetu ladil s ostatními a dal se měnit změnou theme.

Slide 109

Vytvoření nové třídy widgetů

- Definice struktur pro třídu a widget (`GtkWIDGETClass`, `GtkWIDGET`)
- Registrace typu (`gtk_WIDGET_get_type()`)
- Vytvoření instance (`gtk_WIDGET_new()`)
- Inicializační funkce třídy
 - Definice nových signálů
 - Nastavení implicitních handlerů
 - Definice properties
- Inicializační funkce instance

- V GTK+ je pro každý widget samostatný header `gtkWIDGET.h` a C soubor `gtkWIDGET.c`.
- Definují se i makra pro testování typu a přetypování.

Slide 110

Vytvoření nové třídy widgetů (2)

- Zrušení objektu (`dispose`, `destroy`, `finalize`)
- Realizace a mapování (`realize`, `unrealize`, `map`, `unmap`)
- Nastavení velikosti (`size_request`, `size_allocate`)
- Kreslení (`expose_event`)
- Focus a default (kreslit zvýraznění)

- Příklad definice a použití nového widgetu je v ukázkových programech k přednášce: `gtkev.h`, `gtkev.c`, `gtk_ev_usage.c`.

Slide 111

3. Qt

Slide 112

Qt

- Multiplatformní toolkit: Windows, Unix (X11), Mac OS X, embedded Linux
- Objektivě orientovaný, napsaný v C++
- Budeme se zabývat verzí 3.1.
- Doplnkové nástroje:
 - **Qt Designer** – interaktivní návrh uživatelského rozhraní
 - **Qt Linguist** – podpora překladu řetězců v programu
 - **Qt Assistant** – prohlížeč dokumentace
 - **qmake** – generátor souborů `Makefile`
 - **uic (User Interface Compiler)** – překlad XML (Qt Designer) do C++
 - **moc (Meta Object Compiler)** – zpracování rozšířeného C++

- Existují open source language bindings pro Perl (PerlQt) a Python (PyQt), nepodporovaná firmou Trolltech.
 - Verzi 3.1 jsem používal při přípravě přednášky. Nejnovější (v době psaní tohoto slidu, 9.2.2004) je verze 3.3.0.
 - Qt Designer umožňuje interaktivně rozvrhnout widgety uživatelského rozhraní. Výsledek se uloží ve formátu XML a z něho se pomocí utility `uic` vygeneruje kód v C++.
 - Qt Linguist je nástroj pro podporu lokalizace programů. Spolu s utilitami `lupdate` a `lrelease` slouží pro extrahování označených řetězců ze zdrojových souborů, jejich překlad a generování katalogů přeložených řetězců.
 - Utilita `qmake` generuje `Makefile` z *project file* – souboru popisujícího, které zdrojové soubory jsou potřeba pro překlad programu.
 - Qt rozšiřuje C++ o vlastní mechanismy:
 - komunikace mezi objekty (signály a sloty)
 - dynamická identifikace typů
 - dynamické properties (pojmenované hodnoty, které lze číst a měnit)
- Meta Object Compiler ze zdrojového kódu s konstrukcemi Qt generuje normální C++.
- Pro skriptování v rámci Qt aplikací slouží *QSA (Qt Script for Applications)*, založený na jazyce ECMAScript.

Hello World

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

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QPushButton quit("Quit", 0);
    quit.resize(75, 30);
    QObject::connect(&quit, SIGNAL(clicked()),
                    &app, SLOT(quit()));
    app.setMainWidget(&quit);
    quit.show();
    return app.exec();
}
```

- Triviální program, který vytvoří okno obsahující tlačítko s nápisem „Quit“. Po kliknutí na tlačítko program skončí.
- Qt používá oddělené hlavičkové soubory pro jednotlivé třídy. Je potřeba vždy includovat headery pro třídy, které používáme.
- Každý program v Qt má jednu instanci třídy `QApplication`.
- Konstruktor `QApplication` odebere z argumentů ty, které jsou určeny pro Qt (např. `-style`, `-display`, `-title`, `-geometry`).
- Konstruktor tlačítka `QPushButton` definuje nápis na tlačítku a rodičovský widget. Rodičovský widget 0 znamená, že tlačítko je top-level okno.
- Metoda `quit.resize()` nastaví velikost tlačítka (navíc se přidá rámeček window manageru). Kdyby se nevolala, tlačítko bude mít svou „přirozenou“ velikost.
- Statická metoda `QObject::connect()` Připojí *signál* `clicked` generovaný tlačítkem na *slot* `quit` aplikace. To způsobí, že stisk tlačítka ukončí cyklus zpracování událostí.
- Metoda `app.setMainWidget()` nastaví tlačítko jako hlavní widget aplikace. Tím zajistí, že při zrušení tlačítka (např. zavíracím tlačítkem na rámečku okna) se ukončí cyklus zpracování událostí.
- Metoda `show()` zobrazí tlačítko. Na rozdíl od GTK+ `show()` zobrazí i případné potomky widgetu.
- Metoda `app.exec()` spustí zpracování událostí.

Slide 114

Utilita `qmake`

- Automaticky vygeneruje `Makefile` pro překlad a slinkování aplikace.
- Stará se i o správné volání dalších utilit (především `moc`) při překladu.
- Je třeba nastavit environmentové proměnné `QTDIR` (adresář, kde je nainstalované Qt, např. `/usr/X11R6`) a `QMAKESPEC` (kombinace platformy a kompilátoru, např. `freebsd-g++`).
- Základní postup použití:
 1. Uložit zdrojové soubory programu do adresáře pojmenovaného stejně jako program (např. `hello`).
 2. Vytvořit soubor projektu `hello.pro` příkazem `qmake -project`.
 3. Podle potřeby upravit projektový soubor.
 4. Vygenerovat `Makefile` příkazem `qmake`.
 5. Spustit `make`.

- Samozřejmě lze napsat `Makefile` ručně, ale použití `qmake` je pohodlnější.
- Program `qmake` otestuje, které hlavičkové soubory používají rozšíření C++ definovaná v Qt a zajistí, že `make` na ně bude pouštět Meta Object Compiler.

Slide 115

Signály a sloty

- Každá třída odvozená z `QObject` má definovanou množinu signálů a slotů. Signály vypadají jako deklarace metod, mohou mít parametry, ale musí mít vždy návratový typ `void`.
- Objekt vygeneruje **signál**, když se s ním něco zajímavého stane, pomocí `emit signál (argumenty)`.
- **Slot** je metoda, která může být připojená k signálu.
`QObject::connect(odesílatel, SIGNAL(signál),
 příjemce, SLOT(slot));`
Musí odpovídat typy signálu a slotu.
- Po emitování signálu se provedou všechny sloty, které jsou k němu připojeny. Teprve pak se vrátí volání `emit`.

- Signály a sloty fungují podobně jako signály a handlers signálů v GTK+. Navíc Qt poskytuje typovou kontrolu (v době běhu programu).
- Je to mechanismus *synchronní* komunikace mezi objekty.
- Sloty připojené k jednomu signálu se volají v nedefinovaném pořadí.
- Návratová hodnota metody `QObject::connect()` indikuje, zda se připojení povedlo.
- Slot lze odpojit pomocí metody `QObject::disconnect()`.
- Při emisi signálu lze vynechat `emit`, protože toto makro expanduje na prázdnou hodnotu a existuje jen proto, aby se zdůraznilo, že se nevolá běžná metoda, ale emituje se signál.

Slide 116

Události

- Odvozené z třídy `QEvent`
- Generované okenním systémem (`QMouseEvent`), časovačem (`QTimerEvent`), aktivitou na socketu (interní událost pro implementaci `QSocketNotifier`), nebo aplikací (`QCustomEvent`)
- Pro každý typ události (`QPaintEvent`) existuje specifický handler – virtuální funkce `QWidget::paintEvent()`.
- Handler pro určitý typ události se volá z obecného handleru (pro všechny události) `QObject::event()`.
- Objekt se může funkcí `QObject::installEventFilter` zaregistrovat jako **event filter** (filtr událostí) pro jiný objekt. Filtr pak dostává všechny události určené pro cílový objekt dříve než on.

- Události a signály jsou dva oddělené mechanismy, události se nepřekládají na signály jako v GTK+.
- Události se dají používat pro *synchronní* (`QApplication::sendEvent()`) i *asynchronní* (`QApplication::postEvent()`) komunikaci mezi objekty.
- Handler a filtry vrací `TRUE`, jestliže je událost zpracovaná, a `FALSE`, pokud má zpracování dále pokračovat.
- Specifické handlers pro jednotlivé typy událostí ve třídě `QWidget` mají návratový typ `void`. Události z klávesnice a myši se při neakceptování widgetem propagují do rodičovského widgetu. Příslušné třídy událostí (`QKeyEvent`, `QMouseEvent`) proto obsahují metody `accept()` a `ignore()`. Implicitně je událost akceptovaná.
- Event filter se odregistruje voláním `QObject::removeEventFilter()`.
- Instalací filtru na `QApplication` lze filtrovat všechny události v programu.

Slide 117

Definice třídy widgetů

```
class LCDRange : public QVBox {
    Q_OBJECT
public:
    LCDRange(QWidget *parent = 0, const char *name = 0);
    int value() const { return slider->value(); }
    public slots:
        void setValue(int value) { slider->setValue(value); }
    signals:
        void valueChanged(int);
private:
    QSlider *slider;
};
```

- Toto je widget z tutoriálu Qt. Skládá se ze slideru (obdoba `GtkScale`) a zobrazení hodnoty nastavené na slideru pomocí čísla ve stylu LCD.
- Widget je potomkem `QVBox` (to je obdoba `GtkVBox`).
- Nový widget se definuje odvozením z vhodné báze třídy widgetů.
- Aby správně fungovalo vše, co poskytuje Meta Object System, musí odvozená třída v `private` sekci obsahovat makro `Q_OBJECT`.
- Třidu je nutné definovat v hlavičkovém souboru (třidu definovanou v souboru `.cc` Meta Object Compiler nezpracuje). Obvyklá praxe je vytvořit samostatný hlavičkový soubor pro každou třídu widgetů. Obsah tohoto slidu bude v souboru `lcdrange.h`.
- Ve třídě se definují signály a sloty.
- Sloty lze volat i přímo jako libovolné jiné metody.
- U signálů se deklaruje pouze hlavička metody, tělo vygeneruje `moc`.
- Zavoláním signálu se tento signál emituje, tudíž se zavolají všechny k němu připojené sloty.
- Signály se vždy generují jako `protected` metody.
- Sloty mohou být specifikované jako `public`, `protected`, i `private`.

Definice třídy widgetů (2)

```
LCDRange::LCDRange(QWidget *parent, const char *name) :
    QVBox(parent, name)
{
    QLCDNumber *lcd = new QLCDNumber(2, this, "lcd");
    slider = new QSlider(Horizontal, this, "slider");
    slider->setRange(0, 99);
    slider->setValue(0);
    connect(slider, SIGNAL(valueChanged(int)),
            lcd, SLOT(display(int)));
    connect(slider, SIGNAL(valueChanged(int)),
            SIGNAL(valueChanged(int)));
}
```

- Zde je dokončení příkladu z předchozího slidu.
- Obsah tohoto slidu bude v souboru `lcdrange.cc` (nebo `lcdrange.cpp`).
- Widget obsahuje dva synovské widgety `QLCDNumber` a `QSlider` s rozsahem 0–99 a počáteční hodnotou 0.
- První `connect()` zajistí, že při změně hodnoty slideru se aktualizuje LCD číslo.
- Druhý `connect()` ilustruje možnost řetězení signálů. Při napojení jednoho signálu na druhý způsobí první signál emisi druhého.
- Druhý `connect()` se třemi parametry je ekvivalentní volání

```
connect(slider, SIGNAL(valueChanged(int)),
        this, SIGNAL(valueChanged(int)));
```

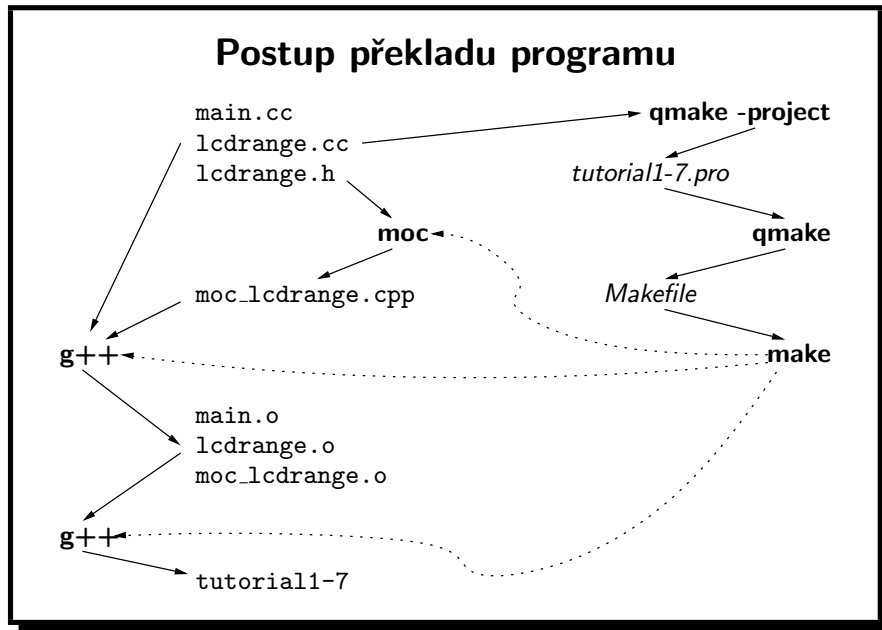
Slide 119

Meta Object Compiler (moc)

- Čte hlavičkový soubor s definicí třídy odvozené z `QObject`.
- Třída musí obsahovat volání makra `Q_OBJECT`.
- Třída dále může obsahovat deklarace signálů, slotů a properties.
- Meta Object Compiler vygeneruje pro třídu soubor `moc_*.cpp`, který obsahuje definice metod implementujících signály a **meta-objektu**.
- Pro každou třídu existuje jeden meta-objekt typu `QMetaObject` (přístupný pomocí virtuální metody `QObject::metaObject()`).
- Meta-objekt popisuje vlastnosti třídy, tj. jméno třídy, jméno bazové třídy, odkaz na meta-objekt bazové třídy, seznamy jmen slotů, signálů a properties.

- Makro `Q_OBJECT` expanduje na deklarace metod, které je nutné předefinovat v každé třídě, např. `metaObject()`.
- Properties se definují makrem `Q_PROPERTY` a v odvozené třídě lze měnit jejich vlastnosti pomocí `Q_OVERRIDE`.
- Informace z meta-objektu používá např. Qt Designer nebo skriptovací engine QSA.

Slide 120



- Na obrázku je znázorněn postup od zdrojových souborů ke spustitelnému programu, včetně zapojení užití qmake a moc.
- Příklad vychází z jednoho programu obsaženého v tutoriálu Qt. Soubory lcdrange.h a lcdrange.cc obsahují definici nové třídy widgetů, main.cc obsahuje funkci main().

Slide 121

Podpora ladění

- Argumenty na příkazové řádce
 - `-nograd` ... program nebude aktivovat grab myši nebo klávesnice
 - `-sync` ... synchronní zpracování požadavků X serverem
- Funkce pro ladicí výpisy
 - `qDebug()` ... výpis ladicího hlášení
 - `qWarning()` ... výpis chybového hlášení
 - `qFatal()` ... výpis chybového hlášení a ukončení programu
 - na `stderr`, jde přesměrovat jinam
- Ladicí makra
 - `Q_ASSERT(b)` ... chyba, když `b` je `FALSE`
 - `Q_CHECK_PTR(p)` ... chyba, když `p` je `0`

- Grab znamená, že žádná jiná aplikace nebude až do ukončení grabu dostávat události z myši nebo z klávesnice. To může způsobit problém, když laděná aplikace aktivuje grab a pak je zastavena debuggerem, např. protože narazí na breakpoint. V tom okamžiku zastavená aplikace drží myš nebo klávesnici a uživatel nemůže manipulovat s debuggerem, aby ji nechal pokračovat.
- Synchronní zpracování je mnohem pomalejší než běžné asynchronní, ale X server okamžitě reportuje případné chyby.
- Lze nastavit handler, který dostane ladicí nebo chybové hlášení a místo výpisu na `stderr` ho např. zobrazí v dialogovém okně.

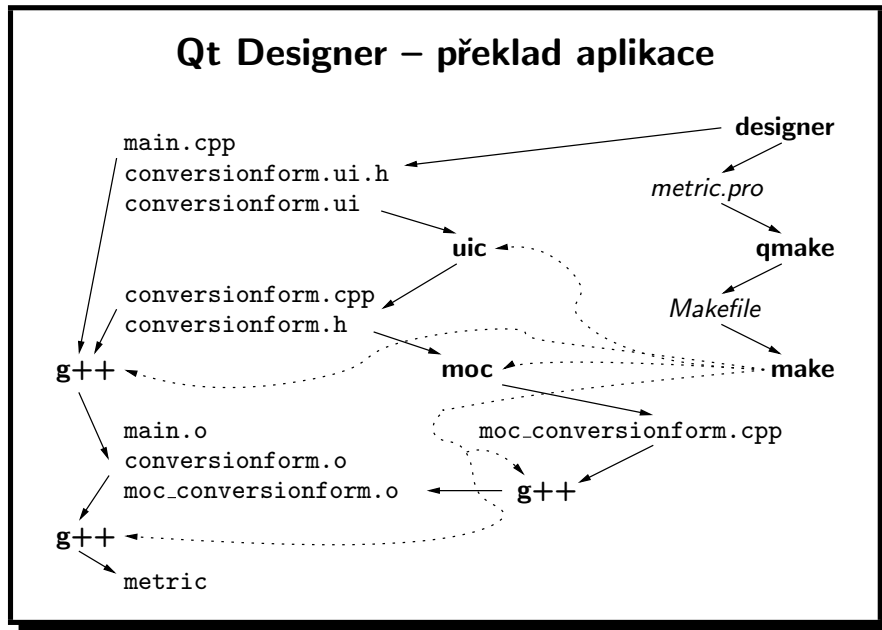
Slide 122

Qt Designer

- Umožňuje grafický návrh aplikace.
- Vytváří soubory:
 - *projekt.pro* ... soubor projektu pro qmake
 - *okno.ui* ... pro každý **formulář** (top-level okno) popis vzhledu okna (struktura a vlastnosti widgetů v okně)
 - *okno.ui.h* ... pro každé top-level okno definice slotů pro třídu reprezentující okno
 - *main.cpp* ... vygenerovaná funkce `main()`, která zobrazí hlavní okno a spustí zpracování událostí
- Program qmake do Makefile přidá instrukce pro generování *okno.h* a *okno.cpp* z *okno.ui* pomocí utility `uic`.

- Nebudeme se programem Qt Designer zabývat detailně. Je to silný nástroj s mnoha funkcemi, který je nejlépe si vyzkoušet. Dokumentace Qt obsahuje i podrobný návod pro Qt Designer včetně tutoriálu.
- Soubory `.ui` se nemusí kompilovat, dají se načítat a dynamicky podle nich vytvářet okna za běhu aplikace pomocí třídy `QWidgetFactory`.

Slide 123



- Na obrázku je nakreslen postup od souborů vytvořených programem Qt Designer až ke spustitelnému programu.
- Příklad vychází z tutoriálu v dokumentaci Qt Designer.
- Zde je v `conversionform.h` a `conversionform.cpp` vygenerovaná definice třídy reprezentující dialogové okno, odvozená z `QDialog`.

Slide 124

Správa paměti

- Objekty se organizují do stromů. Při vzniku objektu se zadává rodič.
- Při zrušení objektu se automaticky zruší všichni jeho potomci.
- Seznam kořenů všech existujících stromů vrací metoda `QObject::objectTrees()`.
- Rodič objektu se dá zjistit pomocí `QObject::parent()`, děti pomocí `QObject::children()`.
- Měnit strom objektů lze metodami `QObject::insertChild()` a `QObject::removeChild()`.
- Stromu objektů obvykle odpovídá stejně uspořádaný strom widgetů, kořenový objekt je top-level okno.
- Strom widgetů mění funkce `QWidget::reparent()`.

- Strom objektů zajišťuje automatické zrušení potomků při zrušení předka.
- Strom widgetů definuje, jak budou vnořená okna na obrazovce.
- Změna stromu objektů nemění strom widgetů a naopak při změně vztahů mezi widgety zůstává nezměněný strom objektů.
- Qt poskytuje šablonu `QGuardedPtr` – ukazatel, který se automaticky nastaví na 0 při zrušení objektu, na který ukazuje.

Slide 125

Sdílení dat

- Při zkopírování objektu se kopíruje pouze ukazatel na data.
- **Implicitní sdílení**
 - Při změně automaticky vytvoří privátní kopii dat.
 - QPixmap, QBrush, QCursor, QFont, QFontInfo, QFontMetrics, QIconSet, QMap, QPalette, QPen, QPicture, QPixmap, QRegion, QRegExp, QString, QStringList, QValueList, QValueStack
- **Explicitní sdílení**
 - Mění se sdílená data, změna se projeví ve všech objektech, které používají stejnou kopii dat.
 - Privátní kopii lze získat metodou detach().
 - QByteArray, QPointArray, QByteArray a všechny další instance šablony QMemArray<type>

- Interně se pro každý blok dat udržuje počet referencí.
- Operátor přiřazení kopíruje pouze ukazatel a zvýší počet referencí.
- Metoda copy() vytvoří objekt s privátní kopií dat s počtem referencí 1.

Umístování widgetů

- Automatické nastavení pozice a velikosti widgetů.
- Layout widgety
 - QHBoxLayout ... widgety vedle sebe, vkládají se zleva doprava.
 - QVBoxLayout ... widgety pod sebou, vkládají se shora dolů.
 - QGridLayout ... widgety ve dvojrozměrné mřížce, zadává se počet sloupců, vkládané widgety postupně zaplňují řádek po řádku.
- Pořadí synovských widgetů na obrazovce odpovídá pořadí jejich vkládání.

```
QGrid *gr = new QGrid(2, parent);  
new QLabel("1", gr); new QLabel("2", gr);  
new QLabel("3", gr); new QLabel("4", gr);  
new QLabel("5", gr);
```

- Tyto widgety jsou jednoduché na používání, ale pokud jsou potřeba složitější operace než pouze postupné vkládání widgetů, je třeba využít některý *manažer geometrie* (geometry manager, potomek třídy QLayout).
- Widgetům QHBoxLayout, QVBoxLayout a QGridLayout odpovídají po řadě manažery geometrie QHBoxLayout, QVBoxLayout a QGridLayout.

- Výsledkem příkladu bude grid:

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |
| 5 | |

Slide 127

Layouts

- Potomci třídy QLayout
- Nejsou to widgety.
- Příklady: QHBoxLayout, QVBoxLayout, QGridLayout
- Poskytují více funkcí než layout widgety.

```
QHBoxLayout *box = new QHBoxLayout(parent);  
box->addWidget(new QLabel("1", parent));  
box->addWidget(new QLabel("2", parent));
```

nebo

```
QHBoxLayout *box = new QHBoxLayout(widget);  
box->setAutoAdd(TRUE);  
new QLabel("1", widget);  
new QLabel("2", widget);
```

- Výsledkem obou příkladů je layout:

| | |
|---|---|
| 1 | 2 |
|---|---|
- Widgety vkládané do layoutu mají jako rodiče *widget*, který obsahuje layout, nikoliv přímo layout.
- Widgety je nutné explicitně vložit do layoutu pomocí metody `addWidget()`.
- Layouty se mohou vnořovat.
- U top-level layoutu (je synem widgetu, ne jiného layoutu) lze metodou `setAutoAdd()` zapnout automatické vkládání widgetů. Následně vytvořené widgety, které mají rodičovský widget shodný s layoutem, budou automaticky vloženy do layoutu.
- Pro layout, jehož rodičem je jiný layout a nikoliv widget, metoda `setAutoAdd()` nefunguje.

Slide 128

Nastavení velikosti widgetů

- Každý widget má svou přirozenou velikost.
- Nejprve se zjistí požadavky widgetů na velikost. Velikost widgetu, který má potomky, závisí na jeho vlastních potřebách a na požadavcích potomků. Přesný způsob skládání potomků do rodičovského widgetu řídí layout.
- Top-level okno dostane přidělenou velikost (v kooperaci s window managerem).
- Velikosti se propagují směrem k potomkům. Distribuce přiděleného místa mezi potomky je opět řízena layoutem.
- Doporučenou a minimální velikost widgetu definují metody `QWidget::sizeHint()` a `QWidget::minimumSizeHint()`. Metoda `QWidget::sizePolicy()` určuje, zda má widget vždy velikost `sizeHint()`, nebo se může zvětšovat či zmenšovat.

- Uvedeným metodám odpovídají stejnojmenné properties ve třídě `QWidget`.
- Při změně hodnot `sizeHint`, `minimumSizeHint` nebo `sizePolicy` se zavoláním metody `QWidget::updateGeometry()` provede přepočítání velikosti.
- Navíc existují ve třídě `QWidget` properties `minimumSize` a `maximumSize`, které definují nepřekročitelné meze velikosti widgetu. Změnit velikost widgetu mimo tyto meze nelze ani přímým nastavením velikosti metodou `QWidget::setGeometry()`.

Slide 129

Kreslení

- Když se má widget překreslit, dostane `QPaintEvent`.
- Kreslení provádí handler `QWidget::paintEvent()`.
- Oblast, která má být nakreslena, vrací `QPaintEvent::region()`.
- Metoda `QWidget::repaint()` zavolá `QWidget::paintEvent()` a tím okamžitě překreslí widget.
- Metoda `QWidget::update()` vloží událost `QPaintEvent` do fronty událostí.
- 2D kreslení zajišťuje třída `QPainter`.
- Pro 3D grafiku je možné použít OpenGL prostřednictvím widgetu `QGLWidget`.

- Stejně jako v Xlib a GTK+, kreslení neprovádí aplikace kdy chce, ale pouze na vnější podnět, když se pomocí události `QPaintEvent` dozví, že je potřeba překreslit widget nebo jeho část.
- Při kreslení do `QGLWidget` se používají přímo funkce OpenGL.

Třída QPainter

- kreslení do `QPaintDevice`, což je widget (`QWidget`), pixmap (rastrový obrázek, `QPixmap`), záznamník kreslicích příkazů (`QPicture`) nebo tiskárna (`QPrinter`)
- parametry kreslení, např. barva a tloušťka čar, barva pro vyplňování, nebo font
- funkce pro kreslení geometrických tvarů (čáry, polygony, kruhy, oblouky, Bezierovy křivky)
- psaní textu aktuálně nastaveným fontem
- kreslení obrázků
- transformace soustavy souřadnic: posun, rotace, změna měřítka, zkosení, nastavení transformační matice

- `QPicture` zaznamenává posloupnost volání metod `QPainter`. Zaznamenané funkce umí zopakovat na nějaký objekt `QPainter`, uložit do souboru nebo nahrát ze souboru. Podporuje svůj formát a SVG.

- Transformační matice má tvar
$$\begin{pmatrix} m_{11} & m_{12} & 0 \\ m_{21} & m_{22} & 0 \\ d_x & d_y & 1 \end{pmatrix} .$$

- Výsledkem transformace je:
$$\begin{aligned} x' &= m_{11}x + m_{21}y + d_x \\ y' &= m_{12}x + m_{22}y + d_y \end{aligned} .$$

- Souřadnice zadané při kreslení se nejdříve transformují pomocí transformační matice `QPainter::worldMatrix()` do souřadnic modelu. Model je oříznut podle nastavení `QPainter::window()` (určuje viditelný obdélník v souřadnicích modelu). Nakonec se výsledek umístí na `QPaintDevice` tak, že se `QPainter::window()` (v souřadné soustavě modelu) ztotožní s `QPainter::viewport()` (v souřadné soustavě zařízení).

Slide 131

Příklad kreslení

```
void CannonField::paintEvent(QPaintEvent *)
{
    QPainter p(this);

    p.setBrush(blue);
    p.setPen(NoPen);

    p.translate(0, rect().bottom());
    p.drawPie(QRect(-35, -35, 70, 70), 0, 90 * 16);
    p.rotate(-ang);
    p.drawRect(QRect(33, -4, 15, 8));
}
```

- Příklad pochází z tutoriálu Qt.
- Při vytvoření objektu `QPainter` se zadává `QPaintDevice`, do kterého bude kreslit.
- Následuje nastavení barvy výplně a vypnutí kreslení obrysových čar.
- Počátek soustavy se přesune do levého dolního rohu (osa y stále směřuje dolů).
- V levém dolním rohu se nakreslí kruhová výseč s úhlem 90° .
- Soustava souřadnic se otočí o `ang` stupňů proti směru hodinových ručiček.
- Nakreslí se obdélník podél otočené osy x .
- `QPainter` je lokální proměnná, proto je na konci metody `paintEvent` automaticky zrušen.

Slide 132

Přehled widgetů

- Bázová třída pro všechny widgety: `QWidget`
- Tlačítka: `QPushButton`, `QCheckBox`, `QRadioButton`
- Posouvátka: `QSlider` a `QDial` (nastavení hodnoty), `QScrollBar` (řízení scrollování)
- Statický text: `QLabel`
- Tooltips: `QToolTip`, `QToolTipGroup`
- Progress bar: `QProgressBar`
- Obrázky: `QPixmap`, `QImage`
- Stavový řádek: `QStatusLine`
- Editační řádek: `QLineEdit`

- Widgety jsou zde uváděny ve stejném pořadí, v jakém jsme probírali widgety GTK+.
- Místo samostatné třídy pro toggle button (funguje jako `QCheckBox`, ale kreslí se jako `QPushButton`) má třída `QPushButton` property `toggleButton`.
- Objekt `QLabel` může obsahovat prostý text, rich text (vypadá jako zjednodušené HTML), obrázek, animaci, nebo číslo.
- `QToolTip` je abstraktní třída, nelze vytvářet její instance, většina metod jsou statické. Odvozovat nové třídy od `QToolTip` a vytvářet instance je potřeba, pokud se tooltipy dynamicky mění. `QToolTipGroup` dovoluje vypínat a zapínat skupiny tooltipů a zobrazovat přídatný text ve stavovém řádku.
- `QPixmap` lze použít pro kreslení, data jsou uložena v okenním systému (v X11 na serveru). `QImage` hardwarově nezávislá reprezentace obrázku s přímým přístupem k obrazovým datům (v X11 uloženo v klientovi), podporuje načítání a ukládání do souboru (ve formátech PNG, BMP, XBM, XPM a PNM, volitelně JPEG, MNG a GIF). Převody mezi `QPixmap` a `QImage` jsou pomalé.

Slide 133

Přehled widgetů (2)

- Spin button (editační řádek s číslem a šipky pro změnu hodnoty): `QSpinBox`
- Combo box (tlačítko nebo editační řádek se seznamem možných hodnot): `QComboBox`
- Seznam vybíratelných read-only položek: `QListBox`
- Vícesloupcový seznam nebo strom položek: `QListView`
- Menu: `QMenuBar`, `QPopupMenu`
- Zobrazení a editace textu: `QTextBrowser`, `QTextEdit`
- Skupina widgetů v rámečku: `QGroupBox`, `QButtonGroup`
- Okno s panely oddělenými posuvnou rozdělovací lištou: `QSplitter`

- `QPopupMenu` je buď připojené k `QMenuBar`, nebo je to samostatné popup menu.
- `QTextBrowser` zobrazuje rich text a umožňuje hypertextovou navigaci.
- `QGroupBox` je orámovaná skupina widgetů uspořádaných do řádků a sloupců. Odvozená třída `QButtonGroup` slouží speciálně pro vkládání tlačítek, každé vložené tlačítko má unikátní identifikátor.

Slide 134

Přehled widgetů (3)

- Posuvná oblast: `QScrollView`
- Toolbar: `QToolBar`
- Kolekce přepínatelných stránek se záložkami: `QTabWidget`
- Hlavní okno aplikace: `QMainWindow`
- Editace data a času: `QDateEdit`, `QTimeEdit`, `QDateTimeEdit`
- Prohlížení a editace databáze: `QDataBrowser`, `QDataTable`, `QDataView`
- Standardní dialogy: `QColorDialog`, `QFileDialog`, `QFontDialog`, `QInputDialog`, `QMessageBox`

- `QToolBar` je potomkem `QDockWidget` a dá se tedy odtrhnout do samostatného top-level okna a zase připojit na původní místo.
- `QMainWindow` obsahuje pruh menu, dokovací oblasti (např. pro toolbary) a stavový řádek.
- K databázovým widgetům existují další třídy pro práci s SQL databázemi.
- Uvedený seznam widgetů není zdaleka vyčerpávající.
- `QColorDialog` – výběr barvy
- `QFileDialog` – výběr jména souboru
- `QFontDialog` – výběr fontu
- `QInputDialog` – modální dialog pro zadání jedné hodnoty (řetězec, číslo, výběr jedné z několika možných hodnot)
- `QMessageBox` – zobrazení krátké zprávy, ikony a několika tlačítek

Slide 135

Dialogy

- Třída `QDialog`
 - Slot `QDialog::showExtension()` umožňuje zapínat přídavné položky dialogu nastavené pomocí `QDialog::setExtension()`.
 - **Nemodální dialog** se zobrazí metodou `show()` a zpracování událostí probíhá normálně dál pro celou aplikaci.
 - **Modální dialog**
 - Spustí se metodou `QDialog::exec()`.
 - Metoda `exec()` vrací návratový kód dialogu.
 - Ukončení dialogu a nastavení návratového kódu provádí sloty `QDialog::accept()`, `QDialog::reject()` a `QDialog::done()`.
-
- Dialog může buď zobrazovat jen základní sadu položek, nebo všechny položky. Přepínání mezi režimy se obvykle provádí připojením signálu `QPushButton::toggled()` na slot `QDialog::showExtension()`.
 - Obvykle se připojuje signál `clicked()` tlačítka „OK“ na slot `QDialog::accept()` a signál `clicked()` tlačítka „Cancel“ na slot `QDialog::reject()`.

Řetězce

- `QString`
 - reprezentace Unicode textu
 - implicitní sdílení
 - konverze interní reprezentace ↔ ASCII, lokální 8bitové kódování a UTF-8
 - metody pro porovnávání, změny, hledání (včetně regulárních výrazů)
- `QString`
 - abstrakce Cčkových řetězců ukončených nulou
 - explicitní sdílení
 - podobná sada operací jako `QString`, ale často méně efektivní implementace

Slide 136

- Když se v Qt zadávají řetězce jako parametry, obvykle se používá `QString`.

Slide 137

Kontejnery, Qt Template Library (QTL)

- obdoba STL
- kontejnerové šablony
 - QMap ... mapování klíč→hodnota
 - QList ... obousměrný spojový seznam
 - QValueStack ... zásobník
 - QVector ... dynamické pole
 - kontejnery, které neukládají přímo hodnoty, ale ukazatele na ně: QMap, QList, QQueue, QStack, QVector
- iterátory: QMapIterator, QListIterator, QMapIterator...
- algoritmy: qHeapSort(), qCount(), qFind()...

- QMap odpovídá STL šabloně map.
- QList odpovídá STL šabloně list.
- QVector odpovídá STL šabloně vector.
- V programu lze podle potřeby míchat STL a QTL.

Časovače

- Vytvoření
`QTimer *t = new QTimer(parent);`
- Nastavení akce při vypršení času
`connect(t, SIGNAL(timeout()), parent, SLOT(timeout()));`
- Nastartování periodického časovače
`t->start(2000);`
- Nastartování jednorázového časovače
`t->start(2000, TRUE);`
- Zastavení
`t->stop();`

Slide 138

- Interval časovače se zadává v milisekundách, ale skutečné rozlišení závisí na operačním systému.
- S časovači se dá pracovat také předefinováním handleru `QObject::timerEvent()` a metod `QObject::startTimer()`, `QObject::killTimer()`.

Slide 139

I/O, síť, procesy

- `QFile` ... čtení a zápis souborů
- `QDir` ... procházení adresářů
- `QFileInfo` ... na platformě nezávislé testování přístupových práv a časů souborů
- `QIODevice` ... na platformě nezávislé low-level API pro sockety
- `QSocket` ... TCP spojení
- `QSocketNotifier` ... obdoba `select()`, generuje signál `activated()` při detekci události na socketu
- `QDns` ... asynchronní DNS lookup
- `QFtp`, `QHttp` ... protokoly FTP, HTTP
- `QProcess` ... spouštění externích programů a komunikace s nimi

- `QSocketNotifier` je vhodné použít pro implementaci neblokujících síťových operací. Blokující operace nejsou použitelné, protože po dobu zablokování v síťové funkci celá aplikace stojí a uživatelské rozhraní nereaguje. Alternativou je přesunutí síťových operací do samostatného vlákna nebo procesu.
- `QFtp` a `QHttp` implementují klientskou stranu protokolů.
- `QProcess` umí spustit externí program s argumenty, asynchronně psát na jeho standardní vstup, číst jeho standardní a chybový výstup, zjistit jeho PID, detekovat ukončení, přečíst návratový kód a násilně běh programu ukončit.
- Pro čtení a zápis textu na `QIODevice` (básová třída pro `QBuffer`, `QFile`, `QSocket` a `QIODevice`) slouží třída `QTextStream`, která poskytuje obvyklé streamové přetížené operátory `operator<<()` a `operator>>()`.

Konfigurace programů

- Třída `QSettings`
- Textové konfigurační soubory v adresáři `~/ .qt/`
- Dvojice klíč (řetězec), hodnota
- Klíč: posloupnost dvou nebo více položek oddělených lomítky
`/Program/Sekce/klic1/klic2` → položka `klic1/klic2` v sekci `[Sekce]` v souboru `~/ .qt/program`
- Hodnota: boolean, integer, double, řetězec, seznam řetězců
- Uložení hodnoty: metoda `writeEntry()`
- Přečtení hodnoty: `readEntry()`
- Zjištění existujících klíčů: `entryList()`, `subkeyList()`
- Zrušení klíče: `removeEntry()`

- Konfigurační soubory se hledají ještě v dalších adresářích (lze ovlivnit pomocí metod `QSettings::insertSearchPath()` a `QSettings::removeSearchPath()`). Později načtené hodnoty mají přednost. Uživatelské nastavení v `~/ .qt/` se obvykle čte jako poslední a proto má nejvyšší prioritu.
- `/Program/klic` → položka `klic` v sekci `[General]` v souboru `~/ .qt/program`
- `QSettings::writeEntry()` je přetížená metoda pro jednotlivé podporované typy.
- Kromě `QSettings::readEntry()` existují ještě `readListEntry()`, `readNumEntry()`, `readDoubleEntry`, `readBoolEntry()`
- Při prohledávání stromu klíčů se rozlišují listy (`entries`) a vnitřní uzly (`subkeys`).

Slide 141

Komunikace mezi procesy

- Třída `QClipboard`
 - jedna instance v aplikaci ... `QApplication::clipboard()`
 - `text()` ... přečte obsah clipboardu jako text
 - `setText()` ... uloží text do clipboardu
 - `data()`, `setData()` ... přečtení/uložení dat v různých formátech
- Drag&drop
 - `drag` ... např. v reakci na událost, uložení dat ve formě instance potomka třídy `QDragObject` a zavolání `QDragObject::drag()`
 - `drop` ... nastavení `QWidget::setAcceptDrops(TRUE)`, předefinování handlerů `QWidget::dragEnterEvent()` a `QWidget::dropEvent()`

- V X11 implementováno pomocí `selections`.
- V metodě `dragEnterEvent()` se testuje, zda data mohou být konvertována do požadovaného formátu, a podle toho se akceptuje událost `event->accept(QTextDrag::canDecode(event));`
- Metoda `QWidget::dropEvent()` zajistí přečtení dat, např. `QTextDrag::decode(event, text);`

Slide 142

Internacionalizace a lokalizace

- Používat `QString` pro texty zobrazované uživateli
- Texty obalit do `tr()`, např.

```
QWidget *w = new QLabel(tr("Full file path:"), dlg);
```
- Používat `QKeySequence` pro definice akcelerátorů
- Používat `QString::arg()` místo `QString::sprintf()`
- Extrahovat řetězce z programu pomocí `lupdate` do souboru `*.ts`
- Přeložit řetězce pomocí **Qt Linguist**
- Vytvořit soubory řetězců (`*.qm`) pro jednotlivé podporované jazyky pomocí `lrelease`
- Vytvořit objekt `QTranslator` a načíst řetězce pomocí `QTranslator::load()` a `QApplication::installTranslator()`

- `QKeySequence` umožňuje zadat akcelerátor pomocí řetězce. Při lokalizaci aplikace do jiného jazyka se snadno změní i akcelerátory.
- `QString::arg()` umožňuje při lokalizaci změnit pořadí argumentů.
- Soubor `.ts` s řetězci je v XML, proto se dá editovat i ručně, není nutné používat Qt Linguist.
- Součástí jména načítaného `.qm` souboru je obvykle kód jazyka. Aktuální jazyk (nastavený v environmentové proměnné `LANG`) zjistí metoda `QTextCodec::locale()`.

Slide 143

4. Xlib

Protokol X

- Klient otevírá stream socket k serveru.
- 4 typy paketů:
 - **request, požadavek** (klient→server) – generuje většina funkcí Xlib, např. nakreslení čáry, vytvoření okna, dotaz na velikost okna
 - **reply, odpověď** (server→klient) – odpověď na požadavek klienta o informace od serveru, např. na dotaz na velikost okna
 - **event, událost** (server→klient) – informace o uživatelském vstupu, změně rozložení oken, nebo zpráva od jiného klienta
 - **error, chyba** (server→klient) – informace o chybě v některém předchozím požadavku

- X používá UNIXový soket `/tmp/.X11-unix/Xd`, kde *d* je číslo displeje, a TCP soket na portu `6000 + d`.
- Shared Memory Extension dovoluje rychlejší přenos dat přes sdílenou paměť, ale klient i server musí být na stejném počítači.
- Request, kterým klient požaduje získání informace od serveru (*round-trip request*) zpomaluje komunikaci. Proto by se takové požadavky měly používat co nejméně. Na požadavky, pro které není potřeba návratová hodnota (např. kreslení) server neodpovídá.
- Zdroje událostí jsou např. stisk klávesy, stisk tlačítka myši, pohyb myši, změna velikosti okna, odkrytí části okna, která vyžaduje překreslení, zrušení okna, komunikace mezi klienty prostřednictvím výběrů.
- Chyby jsou podobné událostem, ale ošetřují se jinak. Chyba se pošle do chybového handleru Xlib. Implicitní chybový handler vypíše chybové hlášení a ukončí program. Chyby přichází asynchronně díky dávkovému předávání požadavků serveru.

Slide 145

Dávková komunikace

- X server má globální frontu událostí, které rozesílá klientům.
- Xlib má lokální frontu událostí přijatých ze serveru, klient si z fronty postupně vybírá události.
- Požadavky se ukládají v Xlib a posílají se najednou, když:
 1. Program zavolá funkci Xlib pro čekání na událost (např. `XNextEvent`), ale nemá v lokální frontě žádnou událost.
 2. Program zavolá funkci Xlib, která vyžaduje odpověď od serveru.
 3. Program explicitně pošle čekající požadavky voláním `XFlush()` nebo `XSync()`.
- Dávkové posílání požadavků se dá vypnout funkcí `XSynchronize()`.

- Dávková komunikace značně urychluje běh aplikace.
- Vypnutí dávkového zpracování se používá pouze pro účely ladění, protože několikanásobně zpomaluje program. Jeho výhodou je, že server oznamuje chyby okamžitě a odpadá zpětné dohledávání požadavku zodpovědného za chybu.

Slide 146

Zdroje (resources), vlastnosti (properties), atomy (atoms)

- **Typy zdrojů:** okno (window), pixmap (pixmap), paleta (colormap), kurzor (cursor), font, grafický kontext (graphics context)
- Spravuje je X server, klienti se na ně odkazují pomocí ID.
- Se zdrojem může manipulovat libovolný klient, který zná jeho ID.
- **Properties** jsou informace asociované s oknem a přístupné všem klientům pracujícím se serverem. Používají se pro předávání informací mezi klienty, např. aplikace pomocí nich předává window manageru požadavky na velikost svého toplevel okna.
- Properties mají řetězcové jméno a numerický identifikátor – **atom**. Aplikace získá ke jménu atom voláním `XInternAtom()`.

- Zdroje spravované X serverem omezují zatížení sítě, protože se nemusí stále přenášet mezi klientem a serverem hodnoty jako pozice a velikosti oken nebo bitmapy znaků ve fontu.
- Při ukončení klienta (odpojení od X serveru) se automaticky uvolní zdroje vlastněné tímto klientem.
- Termín „zdroj“ („resource“) se v X používá ještě v souvislosti s resource managerem, což je něco úplně jiného.
- Používání atomů zabraňuje častému přenášení dlouhých řetězců mezi serverem a klienty.
- V rámci jednoho spuštění serveru je s určitým řetězcem spojen vždy stejný atom a atomy pro různé řetězce jsou různé.

Slide 147

Manažer oken (window manager)

- X klient, který se stará o:
 - řízení toplevel oken aplikací
 - přidělování focusu klávesnice
 - přepínání barevných palet
- Obvykle přidává k toplevel oknům rámečky, pomocí nichž je možné okna přesouvat, zavírat, měnit velikost, nebo ikonizovat.
- Aplikace by měly s window managerem spolupracovat.
- Window managery obvykle umožňují spouštět aplikace a spolupracují se session managerem, případně jsou integrovány do grafického uživatelského prostředí (CDE, GNOME, KDE).

- Window manager je normální klient, pouze používá některé speciální funkce Xlib.
- Window manager přiděluje keyboard focus toplevel oknům jednotlivých aplikací. Předávání focusu mezi okny aplikace je její interní záležitostí.
- Pokud aplikaci nevyhovuje implicitní paleta, může si vytvořit vlastní virtuální paletu a nastavit ji do atributu `colormap` svého toplevel okna. Jestliže potřebuje pro některá další okna jiné palety, nastaví ID těchto oken do property `WM_COLORMAP_WINDOWS` toplevel okna. Window manager se pak stará o přepínání palet, tj. o instalaci virtuálních palet jednotlivých klientů do hardwarové palety (řídí *colormap focus*). Snaží se instalovat co nejvíce palet ze seznamu palet klienta. Aplikace, se kterou uživatel právě pracuje, má instalovanou svou paletu a zobrazuje se ve správných barvách. Okna ostatních programů mohou být zobrazena v nesprávných barvách. Klienti nesmí sami instalovat palety.
- Session manager zajišťuje startování aplikací, uložení jejich stavu při ukončení práce uživatele v X a jejich znovuspouštění při příštím přihlášení uživatele.

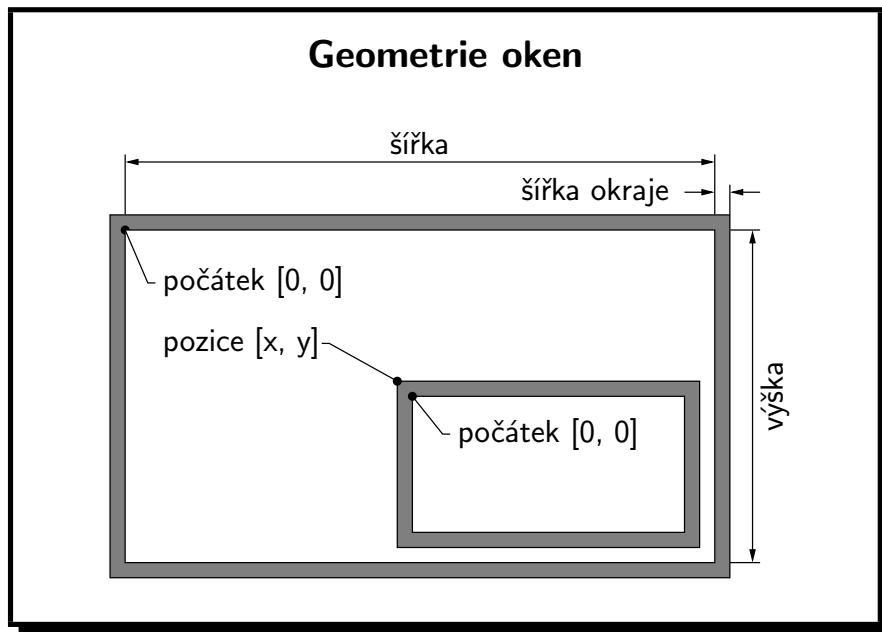
Slide 148

Okno

- Obdélníková oblast obrazovky
- Tvoří stromovou strukturu (na každé obrazovce).
- Root okno pokrývá celou oblast obrazovky.
- Kreslení, přijímání událostí
- Rozšíření pro okna libovolných tvarů
- **Geometrie:**
 - x zleva doprava, y shora dolů
 - souřadnice = pixely
 - okraj okna (nepočítá se do velikosti)
 - pozice okna vzhledem k rodiči
 - velikost bez okraje, pozice včetně okraje

- Root okno existuje po celou dobu běhu serveru a tvoří kořen hierarchie oken.
- Každá obrazovka (screen) má vlastní root okno.
- S oknem může pracovat libovolný klient, který zná jeho ID, nejen ten, který ho vytvořil.

Slide 149



- *Window's origin* (počátek soustavy souřadnic okna) je v levém horním rohu okna *uvnitř okraje*.
- *Pozice okna* jsou souřadnice levého horního rohu *vně okraje* relativně vůči počátku rodičovského okna.
- Okraj okna je neviditelný, když má nulovou šířku.

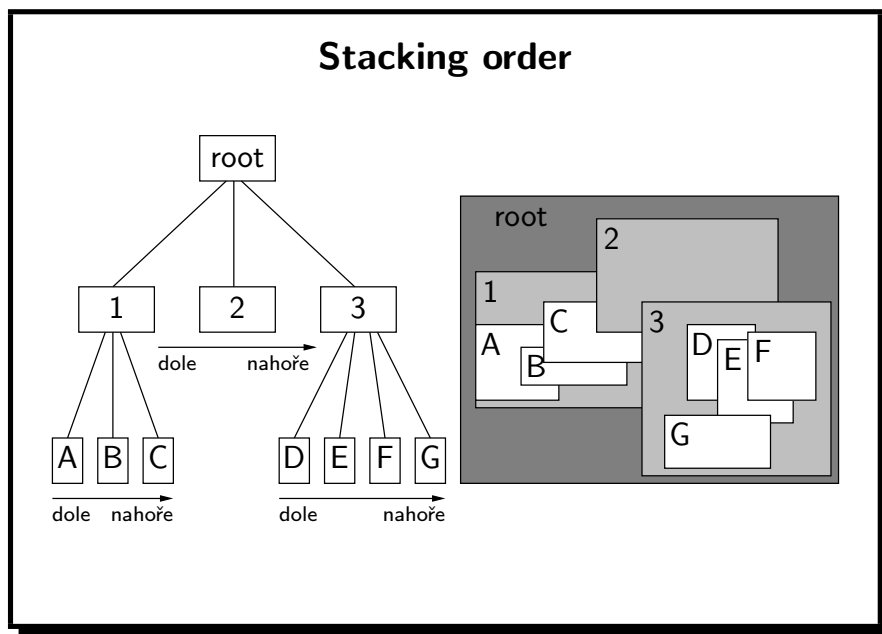
Slide 150

Hierarchie oken

- **Parent (rodič)** – specifikuje se při vytvoření okna
- **Child, subwindow (synovské okno)** – každé okno (kromě root), má rodiče
- **Siblings (sourozenci)** – okna se stejným rodičem
- **Descendants, inferiors (potomci)** – synové okna, jejich synové, atd. až do listů stromu oken
- **Ancestors, superiors (předci)** – rodič okna, jeho rodič, atd. až k root oknu
- **Stacking order** – pořadí mezi sourozenci, určuje, jak se navzájem zakrývají

- Hierarchie oken vzniká při vytváření oken tím, že každé okno musí mít zadaného rodiče. Lze ji měnit, existují funkce pro změnu rodiče (používá je hlavně window manager) a pro změny stacking order (raise, lower, circulate).

Slide 151



- Rodič vždy leží pod všemi potomky.
- Potomci jsou ohraničeni rodičem.

Slide 152

Charakteristiky okna

- Zadávají se při vytvoření okna (`XCreateWindow()`, `XCreateSimpleWindow()`), dají se zjišťovat a měnit i později.
- **Rodič**
- **Souřadný systém** – relativně vůči počátku okna
- **Konfigurace** – pozice, šířka, výška, šířka okraje, stacking order
- **Bitová hloubka** – počet bitů na pixel
- **Visual** – způsob přepočtu hodnoty pixelu na barvu
- **Třída** – `InputOutput` (je vidět na obrazovce, přijímá zprávy), `InputOnly` (není vidět, pouze přijímá zprávy)
- **Atributy** – barvy, gravity, maska událostí, paleta, kurzor, ...

- Pro převod mezi systémy souřadnic oken slouží funkce `XTranslateCoordinates()`.
- I pro input-only okna lze nastavit kurzor.
- Neexistují output-only okna, protože některé zprávy (např. `Expose`) je potřeba přijímat pro každé okno.

Manipulace s charakteristikami okna

- Rodič: `XReparentWindow()`
- Konfigurace: `XGetGeometry`, `XMoveWindow()`, `XResizeWindow()`,
`XMoveResizeWindow()`, `XSetWindowBorderWidth()`,
`XConfigureWindow()`
- Bitová hloubka, visual, třída: nelze měnit
- Atributy: `XGetWindowAttributes()`,
`XChangeWindowAttributes()`

Slide 153

- Zde jsou uvedeny názvy některých funkcí. Xlib jich poskytuje mnohem víc.

Mapování a viditelnost

Aby bylo okno vidět:

- musí být namapováno (`XMapWindow()`)
- musí být namapováni všichni předci
- nesmí být zakryto viditelnými sourozenci nebo sourozenci předků
- musí být vyprázdněn buffer požadavků
- pro top-level okna musí mapování povolit window manager

Klient může předpokládat, že okno je viditelné a dá se do něj kreslit, až po obdržení první události `Expose`.

Slide 154

- Další možnosti mapování: `XMapRaised()` (navíc přesune okno na vrchol stacking order), `XMapSubwindows` (namapuje všechny syny)
- Odmapování: `XUnmapWindow()`, `XUnmapSubwindows()`
- Zakrývání oken závisí na stacking order
- Do vyprázdění bufferu požadavků (čekáním na událost, žádostí o odpověď serveru nebo pomocí `XFlush()`) se požadavky na mapování oken drží v Xlib.

Slide 155

Kořenové (root) okno

- Vytvořeno při inicializaci X serveru
- Předek všech ostatních oken
- InputOutput okno
- Vždy namapované
- Zabírá celou obrazovku, jeho velikost nelze měnit.
- Standardně neposílá události žádnému klientovi (lze změnit).
- Je možné změnit pozadí, kurzor a paletu.

- Root okno tvoří pozadí pracovní plochy X.

Slide 156

Kreslení

- Bitmapová grafika
- Hodnota pixelu se přepočítává na barvu podle palety, způsob přepočtu definuje visual.
- Kreslit se dá po jednotlivých bitových rovinách.
- Kreslí se do okna nebo do pixmapy (dohromady se jim říká **drawables**).
- Parametry pro kreslení se drží na serveru v grafickém kontextu.
- Základní množina grafických primitiv X je poměrně malá, ale doplňují ji různá rozšíření (OpenGL, DPS, Render).

- Grafické kontexty zmenšují množství informací, které musí být obsaženo v požadavcích na kreslení.
- *OpenGL* – knihovna pro 3D grafiku
- *DPS (Display PostScript)* – řízení kreslení pomocí kódu v PostScriptu
- *Render* – 2D grafika, např. antialiasovaný text

Slide 157

Události

- Uživatelský vstup: klávesnice (`XKeyPressEvent`, `XKeyReleaseEvent`), myš (`XButtonPressEvent`, `XMotionNotifyEvent`, `XEnterNotifyEvent`)
- Manipulace s okny: `XCreateNotifyEvent`, `XMapNotifyEvent`, `XConfigureNotifyEvent`
- Kreslení: `XExposeEvent`
- Zpráva od jiného klienta: `XClientMessageEvent`
- Každé okno má pro každého klienta masku událostí (atribut okna `event_mask`), nastavitelnou pomocí `XSelectInput` nebo `XChangeWindowAttributes`.
- Události od klávesnice a myši se z oken, ve kterých nejsou vybrané (v masce událostí), propagují směrem k rodičům.

- Výčet zde uvedených událostí není vyčerpávající.
- Podrobněji se událostem budeme věnovat později.
- `CreateNotify` je vybrána pro rodičovské okno a oznamuje vytvoření synovského okna.
- Událost `Expose` vygeneruje server, když je třeba překreslit část okna.
- Jeden klient druhému může poslat libovolnou událost, nejen `XClientMessageEvent`. V každé události je příznak, který říká, zda ji poslal server nebo klient.
- Když si více klientů vybere stejný typ událostí pro stejné okno, každý dostává kopie událostí. Některé události může mít pro určité okno vybrané jen jeden klient (masky `SubstructureRedirectMask`, `ResizeRedirectMask`, `ButtonPressMask`).

Slide 158

Rozšíření

- Základní X protokol a Xlib jsou neměnné (z důvodu zachování kompatibility), přidávání funkčnosti se dělá pomocí rozšíření.
- Dvě varianty:
 1. Pouze v Xlib (beze změny serveru a protokolu)
 2. V Xlib i v X serveru (s přidáním nových typů zpráv do protokolu)
- Běžně desítky rozšíření najednou aktivních v X serveru. Příklady:
 - 2D grafika ... Render, DPS, XIE
 - 3D grafika ... PEX, GLX (OpenGL)
 - vstup ... XKB, XInputExtension
 - Shape ... libovolné tvary oken
 - MIT-SHM ... komunikace klienta a serveru přes sdílenou paměť
- Existuje definované rozhraní mezi jádrem X a rozšířeními.

- Render – např. antialiasovaný text
- DPS (Display PostScript) – kreslení pomocí kódu v PostScriptu posílaného na server
- XIE (X Image Extension) – zobrazování velkých obrázků, komprimovaný přenos mezi klientem a serverem
- XKB (X Keyboard Extension) – vylepšený mechanismus definice mapy klávesnice
- X Input Extension – podpora jiných vstupních zařízení než klávesnice a myši (např. tablet)

Xlib Hello World

- Připojení k X serveru (`XOpenDisplay()`)
- Vytvoření okna (`XCreateSimpleWindow()`)
- Vytvoření ikony (`XCreateBitmapFromData()`)
- Nastavení vlastností pro window manager (`XSetWMProperties()`), např. titulek okna a ikony, pixmapa ikony, minimální velikost okna
- Výběr typů událostí, které se budou zpracovávat (`XSelectInput()`)
- Načtení fontu (`XLoadQueryFont()`)
- Vytvoření grafického kontextu (`XCreateGC()`)
- Nastavení parametrů grafického kontextu (`XSetFont()`), (`XSetForeground()`, `XSetLineAttributes()`, `XSetDashes()`)

- Po připojení k serveru lze zjistit různé informace o serveru, např. rozlišení obrazovky (`DisplayWidth()`, `DisplayHeight()`).
- `XCreateSimpleWindow()` je zjednodušená (s méně parametry) varianta obecné funkce `XCreateWindow()`.
- Window manager využívá standardní properties okna pro získání informací o tom, jak má zacházet s top-level okny.
- Fonty se zadávají jménem (14 částí oddělených znaky '- ', které popisují jednotlivé vlastnosti fontu), nebo aliasem (zkratka za plné jméno).
- Grafický kontext obsahuje parametry používané při kreslení. Aplikace může používat několik grafických kontextů. Při volání funkce pro kreslení se vždy zadává ID grafického kontextu, který se má použít.

Slide 160

Xlib Hello World (2)

- Namapování okna (`XMapWindow()`)
- Cyklus zpracování událostí:
 - čekání na další událost (`XNextEvent()`)
 - `Expose ...` je třeba překreslit část nebo celé okno (`XDrawString()`, `XDrawRectangle()`)
 - `ConfigureNotify ...` reakce na změnu polohy nebo velikosti okna window managerem
 - `ButtonPress`, `KeyPress ...` uvolnění zdrojů, odpojení od serveru a ukončení programu (`XUnloadFont()`, `XFreeGC()`, `XCloseDisplay()`)

- Mapování top-level okna ovlivňuje window manager.
- `XNextEvent()` vrátí další událost, když žádná nečeká na zpracování, pošle obsah bufferu požadavků na server a čeká na příchod události.
- Po odpojení od serveru se automaticky zruší zdroje, které klient na tomto serveru vytvořil, takže `XUnloadFont()` a `XFreeGC()` nejsou nutné. K odpojení dojde automaticky při ukončení programu, ale je dobré volat vždy explicitně `XCloseDisplay()`, aby klient dostal případné chyby, které zatím nebyly poslány ze serveru na klienta.

Slide 161

Atributy oken

- Nastavení: `XCreateWindow()`, `XChangeWindowAttributes()`,
- Přečtení: `XGetWindowAttributes()`
- barva a pixmapapa pozadí
- barva a pixmapapa okraje
- bit gravity, window gravity
- nastavení backing store
- maska událostí, které se budou doručovat klientovi z tohoto okna
- maska událostí, které se nebudou propagovat do rodiče
- potlačení redirekce
- paleta
- kurzor

- Atributy jsou jednou z charakteristik okna. Další charakteristiky jsou rodič, konfigurace, bitová hloubka, visual a třída.
- Pro nastavení atributů se používá struktura `XSetWindowAttributes`.
- Pro čtení atributů se používá struktura `XWindowAttributes`, která obsahuje i další charakteristiky okna: geometrii, visual, třídu, bitovou hloubku, odkaz na obrazovku a root okno, informace, zda je okno namapováno a zda je instalována jeho paleta.

Slide 162

Pozadí a okraj

- Když se část okna stane viditelnou, server automaticky nakreslí pozadí.
- Pozadí a okraj okna mohou buď být jednobarevné nebo obsahovat pixmapu.
- Možné hodnoty `backgroundPixmap`:
 - `None` ... nemá pixmapu na pozadí; použije se `backgroundPixel`, když není barva pozadí nastavená, bude pozadí průhledné (nebude se kreslit)
 - ID pixmasy ... pozadí okna bude vyplněno pixmapou
 - `ParentRelative` ... použije se pixmapu rodiče a počátek vyplňování pixmapou bude v počátku rodičovského okna

- Jakmile se jednou nastaví barva pozadí, nelze pozadí změnit zpět na průhledné.
- Při vyplňování se pixmapu opakuje tolikrát, kolikrát se do okna vejde. Začíná se od počátku okna.
- Pokud má synovské okno stejnou pixmapu na pozadí jako rodič a pozice syna vůči rodiči není násobkem periody vzorku v pixmapě, budou pixmasy rodičovského a synovského okna fázově posunuté.
- `ParentRelative` na rozdíl od nastavení stejného ID pixmasy, jako má rodič, způsobí, že vyplňování začíná od počátku rodičovského okna. Nedochozí k fázovému posunu mezi pozadími rodiče a syna.

Slide 163

Pozadí a okraj (2)

- Okraj okna kreslí server automaticky.
- Možné hodnoty `border_pixmap`:
 - `CopyFromParent` ... zkopíruje se pixmapu rodiče
 - `None` ... okraj neobsahuje pixmapu; když není nastavena ani barva, vezme se barva rodiče
 - ID pixmapy ... okraj okna bude vyplněn pixmapou, začíná se od počátku okna

- Okraj okna je něco jiného, než rámeček, který obvykle kolem top-level okna kreslí window-manager. Ten obvykle mezi top-level okno a root okno vloží další okno, ve kterém kreslí rámeček s ikonami pro manipulaci s oknem.

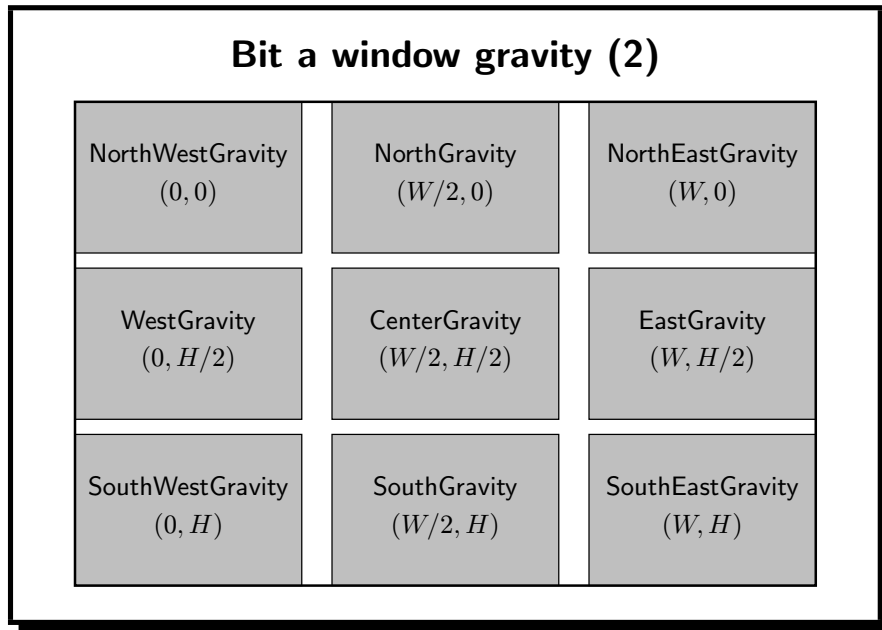
Slide 164

Bit a window gravity

- **Bit gravity**
 - Řídí, kam se posunou existující pixely při změně velikosti okna.
 - Standardní hodnota je `ForgetGravity`, tj. obsah okna se zapomene a musí se překreslit.
 - Zbytek okna kromě pixelů zachovaných díky bit gravity se musí překreslit, server pro ně generuje události `Expose`.
- **Window gravity**
 - Řídí, jak se posunou synovská okna při změně velikosti rodiče.
 - `NorthWestGravity` (standardní) ... nemění se poloha vůči rodiči.
 - `UnmapGravity` ... synovská okna se odmapují.
 - `StaticGravity` ... nemění se poloha vůči root oknu.
 - Ostatní ... okno se posune v závislosti na tom, o kolik rodičovské okno změnilo šířku (W) a výšku (H).

- Server nemusí implementovat bit gravity a může obsah okna zahodit vždy.
- Význam konstant pro bit a window gravity a velikosti posunu pro window gravity jsou na dalším slidu.

Slide 165



- Obrázek ukazuje, kterým směrem se posunuje obsah okna (bit gravity) a synovská okna (window gravity).
- Velikost posunu oken podle window gravity závisí na velikosti změny šířky (W) a výšky (H) rodičovského okna.

Slide 166

Backing store

- Automatické uchovávání obsahu okna při zakrytí nebo odmapování
- Při odkrytí nebo namapování se obsah kopíruje zpět a negenerují se události `Expose` (není třeba překreslovat).
- `NotUseful` (standardní) ... nepoužívat backing store
- `WhenMapped` ... uchovávat obsah pouze, když je okno namapováno
- `Always` ... uchovávat vždy, i při odmapování okna
- Atribut `backing_planes` ... uchovávat pouze některé bitové roviny
- Atribut `backing_pixel` ... hodnota pixelu, která se použije pro roviny neobsažené v `backing_planes`

- Ne každý server podporuje backing store – lze otestovat makrem `DoesBackingStore()`.
- Představuje zátěž pro server.
- Vhodné, pokud klient není schopný překreslovat obsah okna nebo pokud potřebuje kreslit do odmapovaného nebo zakrytého okna.
- Když je zapnuto backing store a okno je větší než rodič, uchovává se celé okno i s oblastmi mimo rodiče.

Slide 167

Save under

- Oblast zakrytá oknem je automaticky uschována před namapováním okna a obnovena při odmapování.
- Používá se pro okna, která jsou zobrazena pouze chvíli, např. menu.
- Makro `DoesSaveUnders()` testuje, zda server podporuje save under.
- `False` ... Při odmapování dostanou odkrytá okna událost `Expose`, pokud jejich obsah nebyl uschován v backing store.
- `True` ... Server obnoví oblasti odkryté odmapováním automaticky, bez generování `Expose`.

- Duální funkce k backing store

Zpracování událostí

- `event_mask`
 - Nastavuje se pomocí `XSelectInput()` nebo `XChangeWindowAttributes()`.
 - Samostatná pro každého klienta
 - Pouze vybrané události se posílají klientovi.
- `do_not_propagate_mask` ... které události od myši a klávesnice neobsažené v `event_mask` se nemají propagovat do rodičovského okna
- `override_redirect` ... požadavky na změnu konfigurace okna nepůjdou přes window manager

Slide 168

- Okno by nemuselo mít masku událostí, protože nezajímavé události může klient jednoduše zahazovat. Ale posíláním zbytečných událostí by se zatěžovalo spojení mezi klientem a serverem.
- Požadavek na změnu konfigurace top-level okna se neprovádí hned, ale pošle se událost window manageru. Ten rozhodne, zda se má požadavek ignorovat nebo provést s původními nebo změněnými parametry.
- Příznak `override_redirect` se obvykle nastavuje pro popup okna, která jsou zobrazena jen chvíli (např. menu) a nechceme, aby s nimi window manager manipuloval.

Slide 169

Paleta a kurzor

- Když se nepředefinují, použijí se hodnoty z rodiče.
- **Barevná paleta**
 - Definuje interpretaci hodnot pixelů v příslušném okně.
 - Pokud hardware nedovolí použití všech palet najednou, window manager řídí, kdy se budou používat které palety.
- **Kurzor**
 - bitmapy a barvy pro popředí a pozadí (masku), hotspot
 - Tvar kurzoru se zadává buď pomocí dvou bitmap, nebo dvou znaků z nějakého fontu.

- Je vhodné počet používaných palet omezit na nezbytné minimum.
- Hotspot je bod, jehož souřadnice se předávají klientům v událostech.
- Znaky pro popředí a pozadí kurzoru se dají vybrat z různých fontů.
- Existuje font `cursor`, který obsahuje standardní kurzory.

Slide 170

Grafický kontext (GC)

- Parametry používané při kreslení
- Uloženy na X serveru, přístupný přes ID
- Aplikace může používat několik GC.
- Vytvoření: `XCreateGC()`
- Nastavení hodnot: `XChangeGC()`, `XCopyGC()`, funkce pro změny jednotlivých parametrů, např. `XSetLineAttributes()`, `XSetFillRule()`, `XSetFunction()`, `XSetForeground()`, ...
- Přečtení hodnot: `XGetGCValues()`
- Zrušení: `XFreeGC()`
- Různí klienti by neměli sdílet stejný GC.

- Lze si vystačit s jedním GC a přenastavovat jeho parametry. Často je rychlejší a pohodlnější připravit si několik GC se všemi potřebnými sadami parametrů a při kreslení si z nich vybírat.
- Při vytváření GC se zadává displej, drawable (GC lze použít pro libovolné drawable se stejnou hloubkou a na stejné obrazovce), hodnoty parametrů v GC a maska určující, které parametry se nastaví podle zadaných hodnot a které dostanou standardní hodnoty.
- `XChangeGC()` nastavuje parametry stejně jako `XCreateGC()`. Funkce `XCopyGC()` kopíruje vybrané parametry z jiného GC.
- Xlib ukládá změny hodnot GC do lokální cache a posílá je na server, teprve když je GC použit.

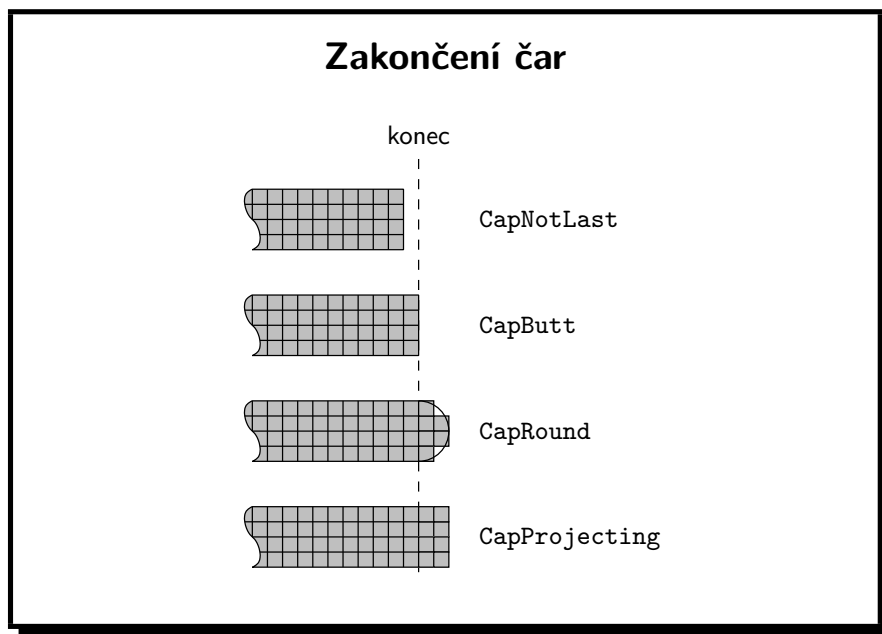
Slide 171

Parametry čar

- Nastavení: `XSetLineAttributes()`, `XSetDashes()`
- `line_width` ... šířka v pixelech
- `line_style`
 - `LineSolid` ... plná čára
 - `LineOnOffDash` ... přerušovaná čára
 - `LineDoubleDash` ... čára kreslená střídavě barvou popředí a pozadí
- `cap_style` ... zakončení čar
- `join_style` ... spojení čar
- `dashes`, `dash_offset` ... vzorek pro přerušované čáry

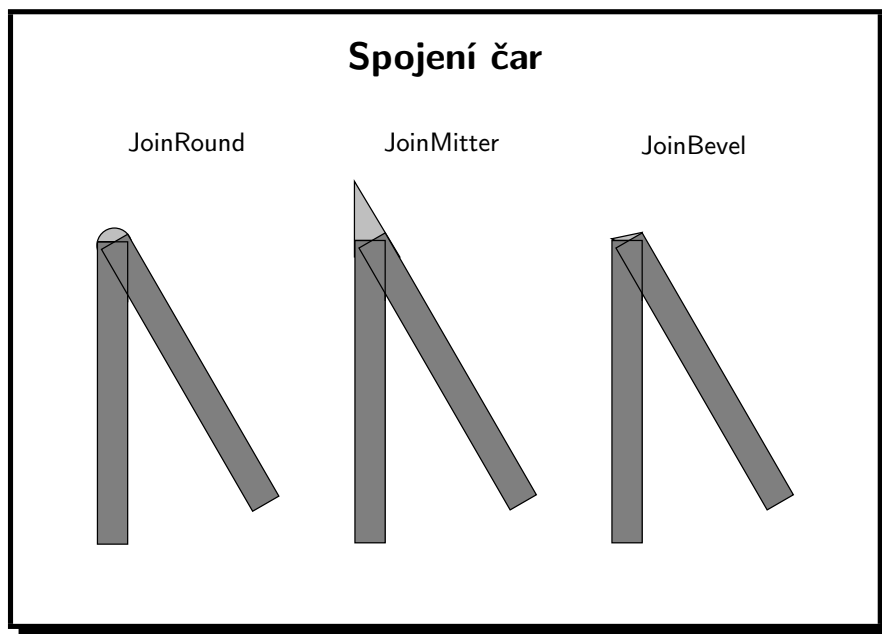
- Šířka 0 znamená tenkou čáru o šířce 1 pixel, často kreslenou jiným rychlejším algoritmem než ostatní čáry. Čáry šířky ≥ 1 vypadají všude stejně, čáry šířky 0 se mohou na různých X serverech lišit.
- Přerušovaná čára (`LineOnOffDash`) má úseky v barvě popředí, mezi nimi se nekreslí, nechává se původní kresba.
- Ve struktuře `XGCValues` lze nastavit `dashes` na jediné číslo udávající délku čárek i mezer. Funkcí `XSetDashes()` se dá nastavit seznam délek, které postupně a cyklicky definují délky jednotlivých čárek a mezer.
- Položka `dash_offset` definuje fázi, tj. jak daleko ve vzorku čárek a mezer čára začíná.

Slide 172



- `CapNotLast` nekreslí koncový bod pouze pro čáry šířky 0 a 1. Pro širší čáry je to totéž jako `CapButt`.

Slide 173



- Řídí spojení čar kreslených jediným grafickým primitivem.

Slide 174

Font

- font ... ID fontu používaného pro kreslení textu
- Nastavení: XSetFont()
- Než klient začne kreslit text, musí font nahrát
 - XLoadFont() ... nahraje font, vrátí ID
 - XLoadQueryFont() ... nahraje font, vrátí informace včetně metrik všech znaků
- XFreeFont(), XUnloadFont() ... uvolní font zadaný pomocí ID nebo XFontStruct *
- Server uvolní font, když už ho žádný klient nepoužívá.

- Vždy je nahraný standardní font.
- Server nahraje každý font do paměti jen jednou.

Slide 175

Vyplňování (fill_rule)

- Pro kreslení vyplněných polygonů (`XFillPolygon()`) a nastavení regionů (`XPolygonRegion()`)
- Nastavení: `XSetFillRule`

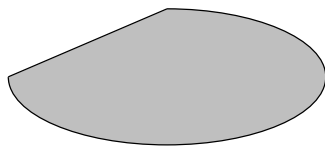
EvenOddRule WindingRule

- Regiony se používají pro ořezávání kresby.
- `EvenOddRule` ... bod je uvnitř, jestliže nekonečně dlouhý paprsek vycházející z bodu protíná obrys v lichém počtu průsečíků.
- `WindingRule` ... bod je uvnitř, jestliže paprsek protíná různý počet levotočivých a pravotočivých segmentů obrysu.

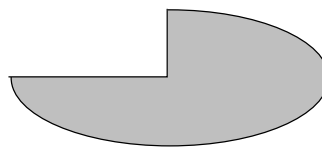
Vyplňování oblouků (arc_mode)

- Pro kreslení vyplněných oblouků (XFillArc(), XFillArcs())
- Nastavení: XSetArcMode

Slide 176



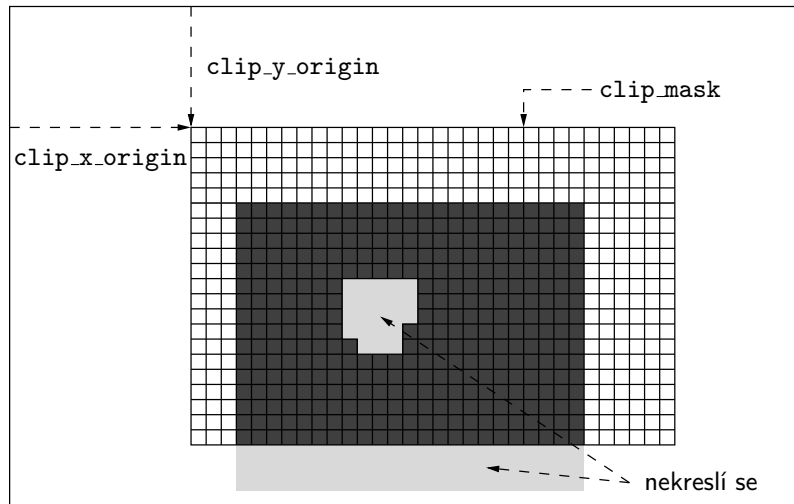
ArcChord



ArcPieSlice

- Při kreslení oblouků se zadává opsaný obdélník, počáteční a koncový úhel.

Ořezávání



Slide 177

- `clip_mask` ... bitmapa určující, které pixely (standardně všechny) budou ovlivněny požadavkem na kreslení
- Nastavení: `XSetClipMask()`, `XSetClipRectangles()`, `XSetRegion()`

Slide 178

Barvy a vzorky

- `fill_style`
 - `FillSolid` ... kreslení barvou popředí (`foreground`)
 - `FillTiled` ... vyplnění pixmapou `tile`
 - `FillStippled` ... vyplnění barvou popředí podle bitmapy `stipple`
 - `FillOpaqueStippled` ... vyplnění barvami popředí a pozadí (`background`) podle bitmapy `stipple`
- `plane_mask` ... určuje, které bitové roviny budou ovlivněny kreslením (standardně všechny).
- `function` ... definuje, jak se skládají bity původní a nové hodnoty pixelu, dá se použít libovolná ze 16 existujících funkcí dvou proměnných s definičním oborem i oborem hodnot `{0,1}`.

- Barva pozadí se používá při:
 - `fill_style` nastaveném na `FillSolid`,
 - použití `XCopyPlane()`,
 - kreslení řetězce pomocí `XDrawImageString()`,
 - při nastavení `line_style` na `LineDoubleDash`.
- Vyplňovací pixmapa musí mít stejnou hloubku jako `drawable`.
- Jedničkové bity ve `stipple` se vyplňují barvou popředí, nulové bity barvou pozadí.
- Vyplňovací pixmapy/bitmappy se opakují tolikrát, kolikrát je potřeba. Počáteční pozice pro vyplňování se zadává relativně vůči počátku `drawable` v položkách `ts_x_origin` a `ts_y_origin`, resp. funkcí `XSetTSOrigin()`.
- Výsledná hodnota pixelu je dána (po bitech) výrazem:
`((src FUNC dst) AND plane_mask) OR (dst AND (NOT plane_mask))`

Slide 179

Graphics exposure, subwindow mode

- `graphics_exposures`
 - řídí posílání událostí při kopírování obsahu drawable (`XCopyArea()`, `XCopyPlane()`)
 - `True` ... standardní hodnota, posílá události `GraphicsExpose`, pokud cílový region nemůže být celý nakreslen, a `NoExpose`, když nakreslen být může
 - `False` ... neposílá při kopírování žádné události
- `subwindow_mode`
 - `ClipByChildren` ... obsah okna je překryt synovskými okny
 - `IncludeInferiors` ... kreslí se i přes synovská okna

- Cíl nelze nakreslit, jestliže část zdrojových dat není dostupná, protože zdrojový region je zakryt nebo není namapován.
- Každá událost `GraphicsExpose` udává jeden obdélník v *cílové oblasti*, který nemohl být nakreslen.
- Tyto události lze vybrat pouze pomocí `graphics_exposures`, ne pomocí `event_mask` nebo `XSelectInput()`.
- `IncludeInferiors` používají window managery pro kreslení obrysu okna při posunu nebo změně velikosti. Obrys se kreslí do root okna.

Slide 180

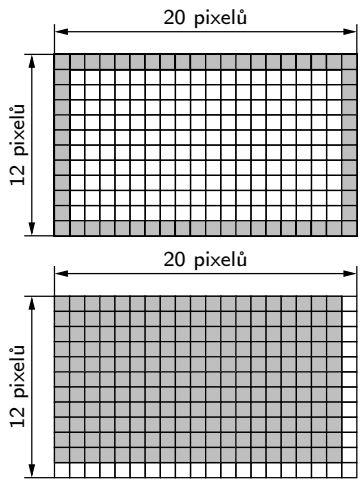
Kreslení

- Body: `XDrawPoint()`
- Úsečky: `XDrawLine()`
- Lomené čáry: `XDrawLines()`, `XDrawSegments()`, `XFillPolygon()`
- Obdélníky: `XDrawRectangle()`, `XDrawRectangles()`,
`XFillRectangle()`, `XFillRectangles()`
- Oblouky: `XDrawArc()`, `XDrawArcs()`, `XFillArc()`, `XFillArcs()`

- `XDrawLines()` kreslí jednu lomenou čáru, `XDrawSegments()` jich může kreslit několik najednou.
- Funkce `XDraw*()` kreslí obrysy, `XFill*()` kreslí vyplněné objekty.
- Počet objektů, které lze nakreslit jedním voláním, závisí na maximální velikosti požadavku X protokolu. Toto maximum (v jednotkách o velikosti 4B) se dá zjistit voláním `XMaxRequestSize()`. Maximum minus 3 je maximální počet bodů. Počet úseček a obdélníků je poloviční, oblouků třetinový.
- Rozměry se udávají v pixelech. Pro přepočítání mezi pixely a milimetry na konkrétní obrazovce slouží makra `DisplayHeight()`, `DisplayHeightMM()`, `DisplayWidth()` a `DisplayWidthMM()`.

Slide 181

Obrys a vyplnění



The diagram consists of two parts, each showing a 20x12 pixel grid. The top part shows a grid with a shaded border, representing the output of `XDrawRectangle`. The bottom part shows a grid that is completely shaded, representing the output of `XFillRectangle`. Both grids have dimensions of 20 pixels by 12 pixels, as indicated by the labels.

```
XDrawRectangle(display,  
drawable, gc, 0, 0, 19, 11);
```

```
XFillRectangle(display,  
drawable, gc, 0, 0, 19, 11);
```

- Obrys je o 1 pixel v každém směru větší než vyplněná oblast.

Slide 182

Mazání a kopírování

- `XClearWindow()`, `XClearArea()` ... smazání obsahu okna nebo obdélníkové oblasti, smazaná oblast je vyplněna barvou nebo pixmapou pozadí
- `XCopyArea()` ... zkopíruje obsah obdélníkové oblasti mezi dvěma drawables, části, pro které nejsou dostupná zdrojová data, jsou vyplněna pozadím a generují se pro ně události `GraphicsExpose` (podle nastavení `graphics_exposures` v GC)
- `XCopyPlane()` ... jedna bitová rovina zdrojové oblasti definuje, které pixely cílové oblasti dostanou barvy pozadí a popředí

- Při mazání se negenerují události `expose`.
- Před překreslením v reakci na událost `Expose` není třeba mazat překreslovanou oblast, protože to server udělá automaticky.
- `XCopyPlane()` se hodí pro překlad bitmapy na pixmapu větší hloubky.

Slide 183

Fonty a text

- Podpora 8- a 16-bitových fontů
- Zjištění dostupných fontů: `XListFonts()`, `XListFontsWithInfo()`
- Načtení fontu: `XLoadFont()`, `XLoadQueryFont()`
- Kreslení textu:
 - `XDrawString()`, `XDrawString16()` ... kreslí pouze znaky barvou popředí
 - `XDrawImageString()`, `XDrawImageString16()` ... kreslí navíc bounding box kolem textu barvou pozadí
 - `XDrawText()`, `XDrawText16()` ... kreslení několika řetězců, pro každý se zadává font a horizontální posun vůči konci předchozího řetězce
- Pozice textu se zadává pomocí začátku **účaří (baseline)**.

- Když chceme horní okraj textu na souřadnici `y`, zadáváme hodnotu `y + ascent`.
- Mezi řádky necháváme mezeru `ascent + descent`.
- V X11R5 bylo přidáno `XFontSet` – sada fontů potřebných pro zobrazení textu v určitém locale.
- Pro zobrazení textu přibyly funkce `Xwc*()` (wide characters typu `wchar_t *`, pevný počet bajtů na znak) a `Xmb*()` (multi-byte characters typu `char *`, proměnlivý počet bajtů na znak).
- V X11R6 došlo k dalšímu zobecnění internacionalizovaného textového výstupu zavedením výstupních metod (output method, output context).

Pojmenování fontů

- **XLFD (X Logical Font Description)**

-foundry-family-weight-slant-setwidth-add_style-pixel_size-point_size-resolution_x-resolution_y-spacing-average_width-charset_registry-charset_encoding
např.

`-misc-fixed-bold-r-normal--13-120-75-75-c-70-iso8859-2`

- Místo dlouhého jména fontu lze zadat alias, např. 5x7 je

`-Misc-Fixed-Medium-R-Normal--7-70-75-75-C-50-IS08859-1`

- Při zadávání jména se dají používat wildcardy, např.

`***-medium-i***-10***-75***-2`

- Velikost 0 znamená škálovatelný font.

- Od verze X11R6 se dá velikost zadat transformační maticí, např.

`***courier-medium-r***-[1e1 ~5.5 ~3.1 18]***-***-***`

- Velikost `point_size` se zadává v desetinách bodu, $1\text{pt} = 1/72.27\text{in}$.
 - Transformační matice umožňuje definovat afinní transformace: změnu velikosti, rotaci a skosení fontu.
 - Transformace se počítají vůči počátku znaku (levý dolní roh).
 - Prvky matice jsou floating-point čísla oddělená mezerou, místo „-“ se používá „~“.
- Matice `[a b c d]` se interpretuje jako
$$\begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$
- Skalární hodnota `N` v `point_size` odpovídá matici `[N/10 0 0 N/10]`.
 - Skalární hodnota `N` v `pixel_size` odpovídá matici `[N*resolution_x/resolution_y 0 0 N]`.
 - Když jsou zadány obě matice, musí platit `pixel_size=point_size*[Sx 0 0 Sy]`, kde `Sx=resolution_x/72.27`, `Sy=resolution_y/72.27`.
 - Když je jedna z matic zadaná jako „0“ nebo „*“, dopočítá se podle druhé.

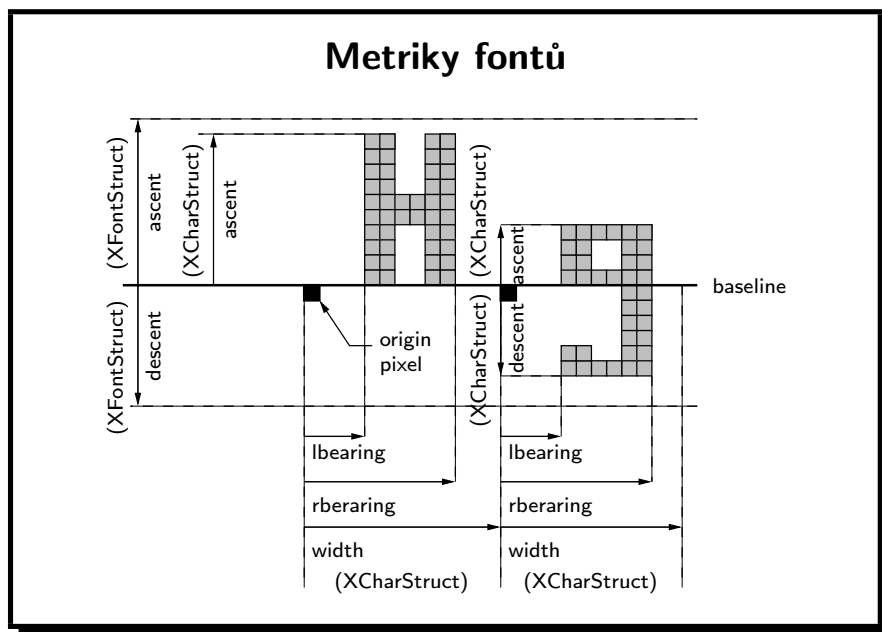
Slide 185

Utility pro fonty

- `xset` ... manipulace s cestou k fontům
- `xfontsel` ... interaktivní výběr fontu
- `xlsfonts` ... výpis seznamu dostupných fontů
- `xfd` ... zobrazení všech znaků z fontu
- `mkfontdir` ... vytvoření seznamu fontů v adresáři
- `xfst` ... font server
- `fslsfonts` ... výpis seznamu fontů poskytovaných font serverem
- `showfont` ... zobrazení fontu načteného z font serveru

- Fonty čte buď přímo X server, nebo je dostává od *font serveru* (`xfst`).
- Fonty jsou *bitmapové* (`pcf` – Portable Compiled Format) nebo *vektorové* (Type1, TrueType).
- X server má cestu k fontům (seznam adresářů s fonty a odkazů na font server).
- V každém adresáři s fonty jsou soubory `fonts.dir` se seznamem fontů (obsahuje mapování mezi jmény fontů a jmény souborů s fonty) a `fonts.alias` (mapování mezi aliasy a plnými jmény fontů).

Slide 186



- Metriky fontu a jednotlivých znaků vrací funkce `XQueryFont()` a `XLoadQueryFont()` ve struktuře `XFontStruct`.

Slide 187

Barvy

- **Frame buffer** ... obrazová paměť, matice obsahující hodnoty pixelů
- **Hodnota pixelu (pixel value)** ... index do palety
- **Paleta (colormap)** ... tabulka pro převod hodnot pixelů na barvy
- **Colorcell** ... jedna položka palety
- Barevný model **RGB** (device-dependent)
- Od X11R5 **device-independent** specifikace barev **Xcms (X Color Management System)**
- Na X serveru je databáze pro převod jmen barev na RGB hodnoty.
- Klienti si alokují položky palety a dostávají hodnoty pixelů.

- Databáze barev na různých serverech může obsahovat pro stejně pojmenovanou barvu různé RGB hodnoty díky *gama korekci*.
- Není to tak, že by klient zadal hodnotu pixelu a barvu, která se má uložit do příslušné položky palety.
- Když se používají jména místo RGB hodnot, je pravděpodobnější, že se dva klienti treří do stejné barvy a položku palety bude možné sdílet.

Xcms

- Xcms device-independent barvy (včetně databáze jmen barev) jsou věci klienta, server používá pouze device-dependent RGB.
- Zadání device-dependent RGB hodnoty:
 - RGB: *red/green/blue*
 - RGBi: *red/green/blue*
 - #RGB
- Zadání device-independent hodnoty:
 - CIEXYZ: *X/Y/Z*
 - CIEuvY: *u/v/Y*
 - CIExyY: *x/y/Y*
 - CIELab: *L/a/b*
 - CIELuv: *L/u/v*
 - TekHVC: *H/V/C*

- Barvu zadanou řetězcem se nejprve snaží interpretovat klient jako specifikaci barvy v jednom z podporovaných barevných prostorů. Když se to nepovede, hledá barvu ve své (na klientovi) databázi barev. Když i to selže, pokouší se barvu interpretovat jako starou (pre-X11R5) formu #RGB, popř. pošle řetězec serveru, který ho hledá ve své databázi barev.
- Funkce jako `XcmsAllocNamedColor()` nejprve převedou zadanou barvu na RGB (device-dependent) a nakonec alokují položku palety pomocí `XAllocColor()`.
- RGB ... každá barevná složka se zadává pomocí 1–4 hexadecadických číslic. Pokud je zadáno méně než 4 číslice, hodnota se bere jako zlomek maximální hodnoty, tj. A není A000, ale 10/15 z FFFF, tedy AAAA.
- RGBi ... každá barevná složka se zadává pomocí floating point čísla v intervalu $\langle 0, 1 \rangle$.
- #RGB ... každá barevná složka se zadává pomocí stejného počtu 1–4 hexadecadických číslic. Kratší formy se interpretují jako nejvyšší bity 16bitové hodnoty, tj. #3a7 je totéž jako #3000a0007000.
- V device-independent barevných prostorech se každá hodnota zadává jako floating point číslo. Platné rozsahy jednotlivých hodnot se liší.

Visual

- Charakterizuje virtuální paletu, která existuje (nebo může být vytvořena) na konkrétní obrazovce.
- Zjištění informací: `XGetVisualInfo()`, `XMatchVisualInfo()`
- Třídy (visual classes):

| typ palety | read/write | read-only |
|-------------------------------|-------------|-------------|
| monochromatická/stupně šedi | GrayScale | StaticGray |
| jeden index pro RGB | PseudoColor | StaticColor |
| samostatné indexy pro R, G, B | DirectColor | TrueColor |

Slide 189

- Jedna obrazovka může mít několik visuals.
- Okna mohou mít virtuální palety. Window manager se stará o instalaci virtuálních palet do hardwarové palety.

Slide 190

Položky palet

- Read-only colorcell
 - Barva nastavena jedním klientem, nelze měnit, lze sdílet mezi klienty
 - Použitelné v měnitelné (DirectColor, PseudoColor, GrayScale) i neměnné (TrueColor, StaticColor, StaticGray) paletě.
- Read/write colorcell
 - Klient, který položku alokoval, může kdykoliv změnit barvu, nelze sdílet.
 - Použitelné pouze v měnitelné paletě.

- Preferováno je použití read-only položek, protože jdou sdílet a a tudíž se celkem ne-spotřebuje tolik položek.

Slide 191

Alokace barev

- Zadává se barva, funkce vrací index v paletě, tj. hodnotu pixelu, která se bude zobrazovat zadanou barvou.
- `XAllocColor()` ... alokuje položku palety podle zadané RGB barvy
- `XAllocNamedColor()` ... barva zadaná řetězcem
- `XcmsAllocColor()`, `XcmsAllocNamedColor()` ... umožňují zadávat i device-independent barvy
- `XAllocColorCells()`, `XAllocColorPlanes()` ... alokuje read/write položky palety
- `XStoreColor()`, `XStoreColors()`, `XStoreNamedColor()` ... nastaví barvy do položek palety

- `XAllocNamedColor()` v X11R5 akceptuje i barvy zadané v sintaxi Xcms.
- Alokuje se vždy zadaná barva, nebo nejbližší barva, kterou lze fyzicky zobrazit.
- Požadavek na alokaci selže, pokud v paletě nejsou volné položky a (pro read-only položky) žádná existující položka neobsahuje požadovanou barvu.

Slide 192

Vytváření a instalace palet

- `XCreateColormap()` ... vytvoření (virtuální) palety
- `XFreeColormap()` ... zrušení (virtuální) palety
- `XSetWindowColormap()` ... nastavení (virtuální) palety pro okno
- `XInstallColormap()` ... používá window manager pro instalaci virtuální palety do hardwarové palety
- událost `ColormapNotify` ... informuje o instalaci, odinstalování a zrušení palety nebo o změně atributu `colormap` okna
- `MinCmapsOfScreen()` ... počet naposled instalovaných palet, které zůstávají současně instalované
- `DefaultColormap()` ... standardní paleta pro obrazovku

- Informace o instalaci, odinstalování a zrušení palety používají aplikace, informace o změně atributu `colormap` používá window manager.
- Počet současně instalovaných palet je často 1.
- Jsou definovány properties popisující typické palety (*standard colormap*). Klient (např. `xstdcmap`), který některou takovou paletu vytvoří, uloží její ID pomocí `XSetRGBColormaps()` a jiný klient se o ní může dozvědět pomocí `XGetRGBColormaps()`. Takto se dají sdílet nejen jednotlivé barvy, ale i celé palety.

Události

- Struktury pro jednotlivé typy událostí (např. XButtonEvent)
- Stejný začátek – struktura pro generickou událost:

```
typedef struct {
    int type; /* typ události (např. ButtonPress) */
    unsigned long serial; /* # posledního požadavku zpracovaného serverem */

    Bool send_event; /* True = posláni XSendEvent() */
    Display *display; /* displej, odkud událost pochází */
    Window window; /* okno, v němž byla událost přijata */
} XAnyEvent;
```

- Union XEvent sdružuje struktury pro všechny typy událostí.
- XSelectInput() ... vybírá, které události se budou klientovi posílat

- Při zpracování události se z XEvent podle event.xany.type vybere příslušná struktura.
- X server a Xlib ukládají události do front v pořadí, ve kterém vznikly.
- Kopie události se posílají všem klientům, kteří mají pro dané okno vybraný její typ.
- Události buď generuje server, (`send_event == False`) nebo si je klienti posílají navzájem (`send_event == True`).
- Události ButtonPress, ButtonRelease, KeyPress, KeyRelease a MotionNotify se podle nastavení atributů okna event_mask a do_not_propagate_mask propagují do rodičů. Zdrojové (source) okno události je nejnižší (a nejmenší) okno v hierarchii událostí, které obsahuje kurzor myši. V položce window v události však bude okno, do kterého se událost propagovala. Distribuce těchto událostí dále závisí na keyboard focus, keyboard grabbing a pointer grabbing.
- Masky pro výběr událostí neodpovídají 1:1 typům událostí:
 - Události MappingNotify, ClientMessage, SelectionClear, SelectionNotify a SelectionRequest jsou vždy vybrané.
 - Např. PointerMotionMask, PointerMotionHintMask, ButtonMotionMask, aj. řídí zasilání událostí MotionNotify.
 - Např. podle SubstructureRedirectMask se posílají události CirculateRequest, ConfigureRequest a MapRequest.

Zpracování událostí

```
XEvent event;  
while(1) {  
    XNextEvent(display, &event);  
    switch(event.type) {  
        case Expose:  
            ...  
        case ButtonPress:  
            ...  
        case MappingNotify:  
            ...  
        default:  
            /* ignorované události */  
    }  
}
```

Slide 194

- Nezájímavé události je nejlepší odfiltrovat pomocí `XSelectInput()` a ve větvi `default` ignorovat už jen události, pro které neexistuje samostatná maska.

Čtení událostí

- `XNextEvent()` ... přečte další událost libovolného typu v libovolném okně
- `XMaskEvent()`, `XWindowEvent()` ... čekání na další událost určitého typu nebo v určitém okně
- `XIfEvent()` ... čekání na další událost splňující obecnou podmínku
- `XCheckMaskEvent()`, `XCheckWindowEvent()`, `XCheckIfEvent()` ... neblokující čtení události
- `XEventsQueued()`, `XPending()`, `XQLength()` ... zjištění počtu událostí ve frontě
- `XPutBackEvent()` ... vrácení události do fronty

Slide 195

- Podmínky v `XIfEvent()`, `XCheckIfEvent()` a `XPeekIfEvent()` se zadávají jako ukazatel na funkci, kterou jsou postupně testovány jednotlivé události ve frontě.
- Funkce pro zjišťování počtu událostí se liší podle toho, jestli posílají na server buffer požadavků.
- Seznam funkcí není vyčerpávající.

Slide 196

Focus

- **Keyboard focus** ovlivňuje distribuci událostí `KeyPress` a `KeyRelease`.
- Vstup z klávesnice dostává pouze okno, které má focus, a jeho potomci. Uvnitř nich funguje propagace událostí normálně. Klávesnicové události vzniklé v jiných oknech jsou doručeny do okna, které má focus.
- Focus přiděluje top-level oknům window manager. Aplikace, jejíž top-level okno má focus, může přidělit focus jeho potomkům.
- `XSetInputFocus()` ... nastaví focus
- `XGetInputFocus()` ... zjistí, které okno má focus
- Když se focus přesune z okna `win1` do `win2`, dostane `win1` událost `FocusOut` a `win2` událost `FocusIn`.

- Toolkit může nechat focus na top-level oknu a uvnitř něho používat pro focus interní mechanismus neviditelný pro ostatní klienty. Takto funguje např. GTK+.

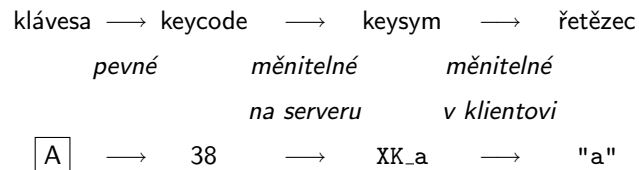
Slide 197

Grab

- Když má některý klient nastavený **grab** na klávesnici nebo myš, pouze tento klient dostává příslušné události.
- `XGrabPointer()`, `XUngrabPointer()`, `XGrabKeyboard()`, `XUngrabKeyboard()` ... nastaví a zruší **aktivní grab** myši a klávesnice.
- `XGrabButton()`, `XUngrabButton()`, `XGrabKey()`, `XUngrabKey()` ... nastaví a zruší **pasivní grab**.
- Mezi `ButtonPress` a `ButtonRelease` je proveden automatický grab, pokud má klient vybrané obě události.
- `XGrabServer()`, `XUngrabServer()` ... monopolizuje X server pro jednoho klienta.

- *Aktivní grab* je proveden ihned.
- *Pasivní grab* znamená, že aktivní grab bude zahájen stiskem nějaké klávesy nebo tlačítka myši.
- Grab má přednost před nastavením fokusu.
- Klient, který má grab, může omezit pohyb kurzoru na některé své okno.
- Grab celého serveru se používá, např. když window manager při zvětšování okna kreslí a maže rámeček pomocí `GXxor`.

Klávesnice



Slide 198

- V `XKeyEvent` je **keycode**.
- Mapování závisí na stavu **modifikátorů** (např. Shift), jejichž stav je v `XKeyEvent.state`.
- Mapování na **keysym** a **řetězec** se dělá funkcí `XLookupString()`.
- `XRebindKeysym()` ... změna mapování keysym na řetězec
- Změna mapování klávesnice: `xmodmap` (standardní obsluha klávesnice), `setxkbmap` (X Keyboard Extension)

- Očíslování kláves (mapování klávesa→keycode) je pevně definováno X serverem.
- Mapování keycode→keysym:
 - definováno globálně, běžní klienti ho typicky nemění
 - mění se pomocí `XChangeKeyboardMapping()`
 - po změně dostane každý klient událost `MappingNotify` a musí aktualizovat svoji kopii klávesové mapy voláním `XRefreshKeyboardMapping()`
- Mapování keysym→řetězec je lokální v každém klientovi, dá se použít např. pro přiřazení často psaných řetězců funkčním klávesám.
- Modifikátory mění funkce `XInsertModifiermapEntry()`, `XDeleteModifiermapEntry()`, `XSetModifierMapping()`.
- XKB poskytuje větší flexibilitu při definování klávesnicové mapy.

Slide 199

Internacionalizace (i18n), lokalizace (l10n)

- **i18n** ... aplikace funguje (beze změny kódu) v různých locale
- **l10n** ... přidání doplňků (databáze pravidel, fonty, klávesové mapy) pro určité locale
- Nastavení locale:
 1. `setlocale(LC_ALL, "")` ... nastaví locale
 2. `XSupportsLocale()` ... otestuje, zda Xlib podporuje nastavené locale
 3. `XSetLocaleModifiers()` ... nastaví (obvykle prázdný) seznam modifikátorů – zřetěžené hodnoty tvaru `@kategorie=hodnota`, k parametru funkce se přidá obsah proměnné prostředí `XMODIFIERS`

- Např. nastavení vstupní metody: `XMODIFIERS="@im=_XWNMO"`
- Informace o locale pro X jsou v adresáři `/usr/X11R6/lib/X11/locale/`.

Slide 200

Internacionalizovaný výstup

- `XFontSet` ... skupina fontů potřebných pro výstup textu v nějakém locale
- `XCreateFontSet()` ... vytvoří `XFontSet`
- Funkce pro práci s **wide-character** (`Xwc*()`) a **multi-byte** (`Xmb*()`) řetězci
- `Xmb/XwcTextEscapement()`, `Xmb/XwcTextExtents()` ... velikost místa zabraného textem
- `Xmb/XwcDrawString()`, `Xmb/XwcDrawImageString()`, `Xmb/XwcDrawText()` ... kreslení textu
- X11R6 zavedlo obecnější mechanismus: výstupní metody a kontexty (**output method, output context**).

- Pro font set existují další funkce na zjišťování informací o jednotlivých fontech (jména, metricky, apod.).
- Pro evropské jazyky (včetně češtiny) stačí typicky nastavit nějaký font v kódování ISO8859-2.
- V X11R6 je interface `XFontSel` implementován jako emulace nad výstupními metodami.

Slide 201

Internacionalizovaný vstup

- Používá komunikaci se vstupní metodou (**input method**). Stav vstupní metody je uložen ve vstupním kontextu (**input context**).
- Vstupní metoda používá:
 - **Status area** je okno, kam IM zobrazuje informace o svém stavu.
 - **Preedit area** zobrazuje text během kompozice znaku.
 - **Auxiliary area** zobrazuje menu a dialogy používané vstupní metodou.
- Styly vstupních metod (umístění oblastí status a preedit):
 - **root-window** ... v samostatném top-level okně
 - **off-the-spot** ... pevné místo v okně klienta
 - **over-the-spot** ... samostatné okno na místě vkládání textu
 - **on-the-spot** ... IM předává data k zobrazení aplikaci

- Každý IC má samostatné oblasti *Preedit* a *Status*.
- *Status area* obsahuje např. indikátory nějakých modifikátorů.
- *Preedit area* se používá pro jazyky (např. asijské), ve kterých se znak postupně skládá z několika jednodušších znaků.
- Oblasti status a preedit poskytuje klient, ale jejich obsah řídí IM. O oblast auxiliary se stará pouze IM.

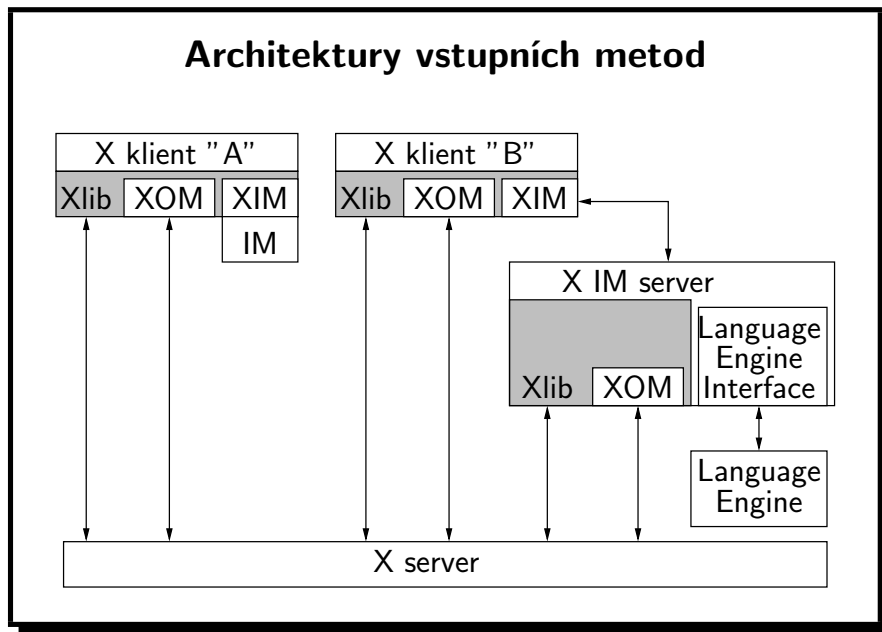
Slide 202

Vstupní metody

- Před zahájením vstupu musí aplikace otevřít vstupní metodu (`XOpenIM()`) a vytvořit vstupní kontext (`XCreateIC()`).
- Každá událost se po přečtení (`XNextEvent()`) nejprve předá vstupní metodě pomocí `XFilterEvent()`.
- Modely zpracování událostí:
 - *front-end* ... IM zachycuje události z X serveru předtím, než se dostanou k aplikaci
 - *back-end* ... IM filtruje události přijaté aplikací dříve, než je aplikace zpracuje
- Metody řízení toku událostí:
 - *static event flow control* ... události jdou vždy přes IM server
 - *dynamic event flow control* ... události obcházejí IM server, kdykoliv to jde

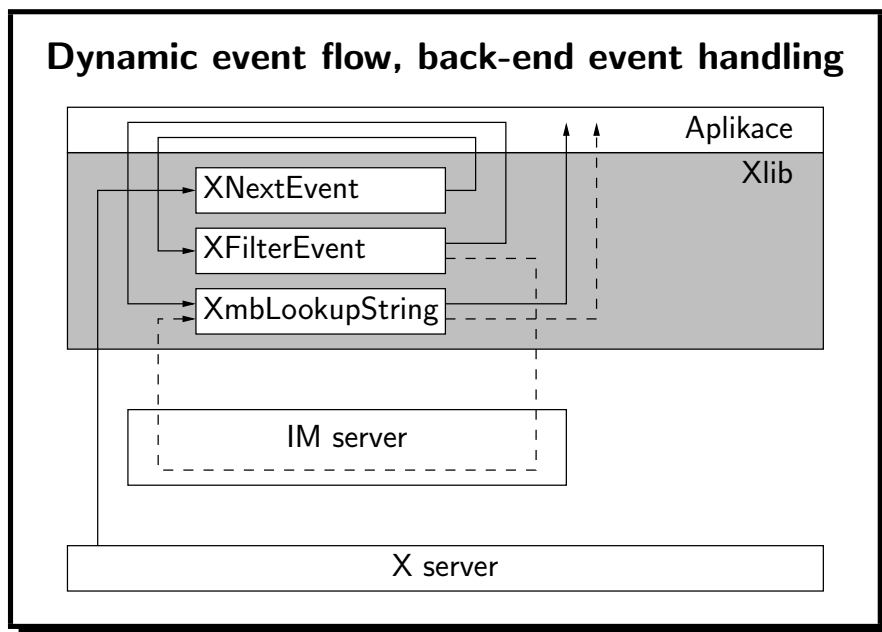
- Vstupní metoda je buď součástí Xlib (např. jednoduché IM pro evropské jazyky), nebo je to samostatně běžící program (*XIM server*, např. pro asijské jazyky).

Slide 203



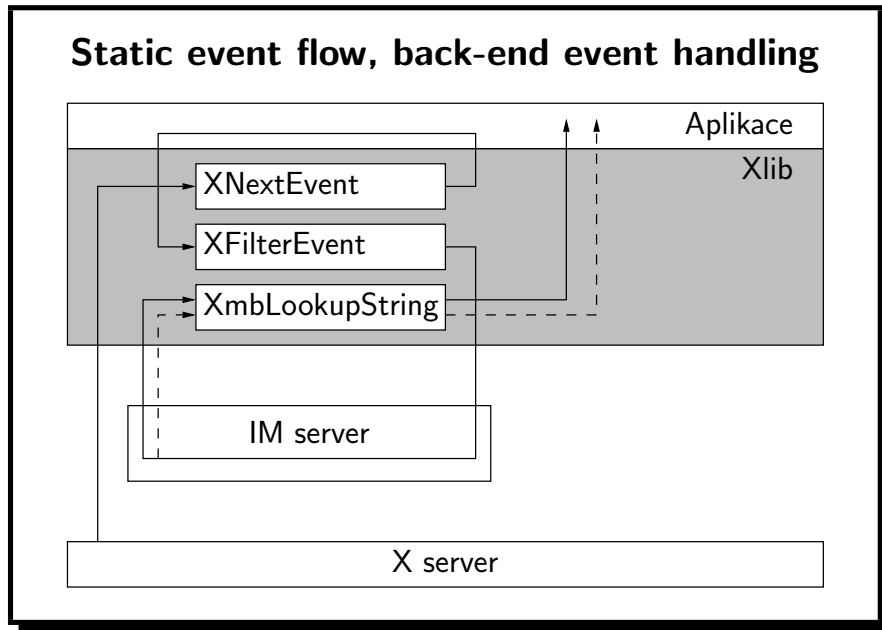
- Klient *A* používá lokální vstupní metodu v Xlib. Klient *B* komunikuje s IM serverem, který dále spolupracuje s language engine (konverze, hledání ve slovníku).

Slide 204



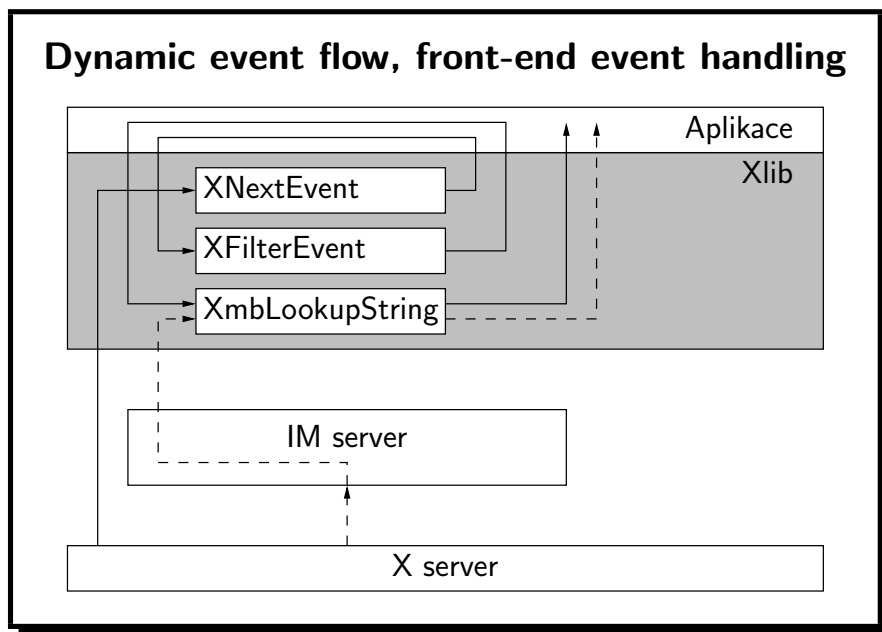
- Přerušovaná čára znázorňuje tok událostí během preeditace.
- Plná čára znázorňuje tok událostí v ostatních případech.

Slide 205



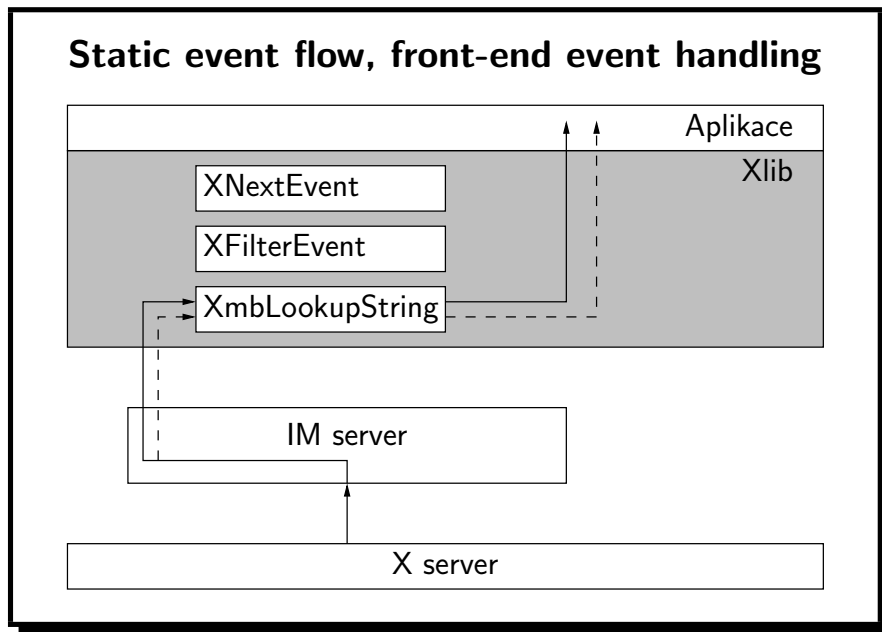
- Přerušovaná čára znázorňuje tok událostí během preeditace.
- Plná čára znázorňuje tok událostí v ostatních případech.

Slide 206



- Přerušovaná čára znázorňuje tok událostí během preeditace.
- Plná čára znázorňuje tok událostí v ostatních případech.

Slide 207



- Přerušovaná čára znázorňuje tok událostí během preeditace.
- Plná čára znázorňuje tok událostí v ostatních případech.

Slide 208

Komunikace mezi klienty

- Pravidla definuje **Inter-Client Communication Conventions Manual (ICCCM)**.
- Přes **properties** – klient nastaví property, ostatní se o tom dozví pomocí události (`PropertyNotify`) a mohou property přečíst.
- Property: $okno \times atom \rightarrow hodnota$ (typ identifikovaný atomem)
- Standardní properties pro komunikaci s window managerem
- **Cut buffers** ... 8 properties asociovaných s root oknem obrazovky 0, klienti se musí předem domluvit na formátu dat
- **Selections** ... obecnější než cut buffers, zajišťují i domluvu na formátu
- `XSendEvent()` ... přímé poslání události mezi klienty

- ICCCM je doporučení, jak se mají klienti chovat. Je na jednotlivých klientech, aby ho dodržovali.
- I klient, který nekomunikuje s ostatními, by měl nastavit hodnoty pro window manager.
- Už dříve jsme poznali použití selections v GTK+.

Slide 209

Properties

- Uložené v X serveru
- `XInternAtom()` ... převod jméno → atom
- `XGetAtomName()` ... převod atom → jméno
- Pro atomy předdefinované v `<X11/Xatom.h>` není třeba volat `XInternAtom()`.
- Properties jsou zrušeny, když je zrušeno okno, ke kterému patří.
- `XChangeProperty()` ... vytvoření/změna property
- `XGetWindowProperty()` ... přečtení property
- `XListProperties()` ... seznam properties okna
- `XDeleteProperty()` ... zrušení property
- Vytvoření, změna a zrušení property generuje `PropertyNotify`.

- Atomy zůstávají definované po celou dobu běhu serveru.
- Properties nejsou zrušeny, když skončí klient, který je nastavil, pokud zároveň není zrušeno příslušné okno (např. jestliže klient nastavil property na cizí okno).

Slide 210

Standardní properties pro window manager

- Nastavení pomocí `XSetWMProperties()`
- Titulek okna
- Titulek ikony
- Příkazový řádek a jméno počítače
- Ikona
- Počáteční pozice ikony
- Pokyny (hints) pro velikost okna
- Počáteční stav (normální nebo ikona)
- Model pro keyboard focus
- Skupina top-level oken jedné aplikace

- Window manager nemusí respektovat nastavené hodnoty.
- Příkazový řádek a jméno počítače jsou určeny pro session manager.
- Ikona je buď pixmap + maska nebo okno.
- Modely keyboard focus:
 - *No Input* (`XWMHints.input == False`) ... klient nepotřebuje vstup z klávesnice
 - *Passive Input* (`XWMHints.input == True`) ... klient nenastavuje sám focus
 - *Locally Active Input* (`XWMHints.input == True`) ... klient nastavuje focus, pouze když už ho některému z jeho oken nastavil window manager
 - *Globally Active Input* (`XWMHints.input == False`) ... klient nastavuje focus, i když ho ještě žádné z jeho oken nemá; měl by to dělat pouze v reakci na události `ButtonPress`, `ButtonRelease`, nebo pasivně grabované `KeyPress` nebo `KeyRelease`
- Skupina top-level oken jednoho klienta může být označena, aby byla spravována dohromady (např. všechna okna ikonizována do jedné společné ikony), společné vlastnosti skupiny definuje vedoucí okno skupiny.
- Dočasná okna (např. dialogy) by měla použít `XSetTransientForHint()` a tím nastavit property `XA_WM_TRANSIENT_FOR` na ID top-level okna aplikace. Window manager pak může pracovat s takovými okny jinak.
- Pro okna, o která se nemá window manager vůbec starat, je třeba nastavit atribut okna `override_redirect`.

Slide 211

Volitelné properties pro window manager

- WM_COLORMAP_WINDOWS ... seznam oken, které používají jinou paletu než top-level okno
- Protokoly (WM_PROTOCOLS):
 - Klienti od window manageru dostávají události ClientMessage
 - WM_TAKE_FOCUS ... pro Locally a Globally Active Input, window manager sděluje, že aplikace si může nastavit focus voláním XSetInputFocus().
 - WM_SAVE_YOURSELF ... oznámení, že klient má uložit svůj stav, protože ho window manager nebo session manager chce ukončit
 - WM_DELETE_WINDOW ... klient dostane žádost o zrušení okna a buď ji ignoruje, nebo okno sám zruší

- Každé okno, které má svou paletu, ji má v atributu colormap.
- Pokud aplikace používá jen jednu paletu (nastavenou pro top-level okno), není třeba definovat WM_COLORMAP_WINDOWS.
- Na WM_SAVE_YOURSELF klient reaguje tak, že uloží svůj stav a nastaví XA_WM_COMMAND (pomocí XSetCommand()) na příkaz, který ho restartuje do stejného stavu. Session manager čeká na PropertyNotify vygenerované změnou XA_WM_COMMAND. Během procesu ukládání není dovolena interakce s uživatelem.
- Klient bez WM_DELETE_WINDOW bude odpojen od serveru, jestliže uživatel prostřednictvím window manageru zruší některé jeho top-level okno.

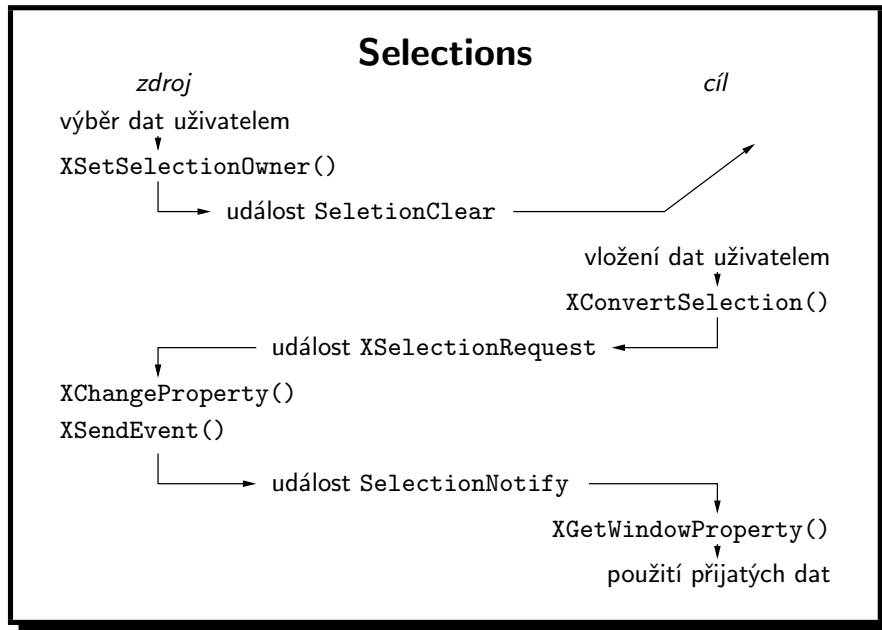
Slide 212

Komunikace window manager → klienti

- `XA_WM_ICON_SIZE` (v root okně) ... povolené velikosti ikon
- `WM_STATE` (v top-level okně) ... pro komunikaci mezi window managerem a session managerem
- Klient se dozví o manipulaci s top-level oknem nastavením masky událostí `StructureNotifyMask`. Na tyto události by neměl reagovat změnou charakteristik, které nastavil window manager.
- `XReconfigureWMWindow()` ... manipulace s top-level okny fungující i pro reparenting window manager. Window manager dostane žádost o změnu, provede ji (příp. pozměněnou) nebo odmítne. Výsledek se klient dozví z události `ConfigureNotify`.

- Podle existence `WM_STATE` se dají poznat top-level okna. Obsahuje stav `WithdrawnState`, `NormalState`, nebo `IconicState`.
- `XConfigureWindow()` selže pro reparenting window manager při změně stacking order.

Slide 213



- Komunikace prostřednictvím výběrů funguje podobně jako v GTK+.
- Lze definovat libovolné množství výběrů, ale obvykle se používá `XA_PRIMARY`, popř. `XA_SECONDARY`.

Slide 214

Selections (2)

- Při přivlastnění výběru dostane předchozí vlastník událost `SelectionClear`.
 - Příjemce dat při volání `XConvertSelection()` udává, do jaké property má vlastník data uložit a požadovaný typ (`target`).
 - Typy dat (cíle, `targets`) se zadávají pomocí atomů.
 - Vlastník uloží data do property (pomocí `XChangeProperty()`).
 - Vlastník pošle příjemci zprávu `SelectionNotify` pomocí `XSendEvent()`. V položce property zprávy je atom property s uloženými daty nebo `None`, pokud výběr nelze do požadovaného typu konvertovat.
-
- `XSetSelectionOwner()` neprovede změnu vlastníka výběru (ale neohlásí žádnou chybu), jestliže ve volání je zadán čas starší než poslední změna vlastníka výběru, nebo čas v budoucnosti (podle hodin serveru).

Selections (3)

- Cíl TARGETS vrací seznam atomů reprezentující podporované cíle.
- Vlastník se může vzdát výběru voláním `XSetSelectionOwner()` s parametrem `None`.
- Velikost výběru je limitována maximální velikostí property.
- Pro velké výběry může vlastník vrátit property `INCR`. Následně vždy příjemce smaže obsah property a vlastník reaguje uložením další části dat. Konec je signalizován daty o velikosti 0. Typ dat je dán typem první části, další části musí mít stejný typ.

Slide 215

- Properties mají omezenou maximální velikost. Není úplně jasné, kolik to je, ale do property by se asi mělo vejít aspoň tolik, kolik lze uložit jedním voláním `XChangeProperty()`.

Slide 216

Cut Buffers

- Properties root okna `XA_CUT_BUFFER0` až `XA_CUT_BUFFER7`
- Klient musí nejdříve zajistit, že všech 8 existuje (přidáním dat délky 0 do všech pomocí `XChangeProperty`).
- Klient, který chce uložit data, nejprve rotuje buffery o +1 voláním `XRotateWindowProperties()` nebo `XRotateBuffers()`.
- Pak uloží data do bufferu 0 voláním `XStoreBytes()`.
- Příjemce dat si je vyzvedne z bufferu 0 funkcí `XFetchBytes()`.
- Na žádost uživatele je možné rotovat buffery zpět o -1.

- Cut buffers jsou properties, proto je při jejich změně generována událost `ChangeNotify`.

Slide 217

Ladicí a administrátorské utility

- `xset` ... nastavení X serveru
- `xdpyinfo` ... informace o displeji
- `xlsclients` ... seznam klientů
- `xlsatoms` ... seznam definovaných atomů
- `xwininfo` ... informace o okně nebo stromu oken
- `xprop` ... zobrazení a nastavení properties
- `xev` ... zobrazování přijatých událostí
- `bitmap` ... editor bitmap
- `xrdb` ... manipulace s resource databází – property
RESOURCE_MANAGER root okna obrazovky 0 nebo
SCREEN_RESOURCES libovolné obrazovky
- `appres` ... vypsání resource databáze aplikace

- `xset` nastavuje autorepeat klávesnice, upozorňovací tóny (bell), screen saver, cestu k fontům a DPMS.
- `xdpyinfo` zobrazuje identifikaci X serveru, maximální velikost požadavku, podporované formáty pixmap, seznamy rozšíření, obrazovek a visuals.
- Resource databáze je seznam konfiguračních informací pro klienta, neplést se server resources.
- Seznam dalších utilit viz `man X`.

Slide 218

Konfigurace klientů

- **resources** ... textové konfigurační soubory a resource databáze X serveru
- **standardní argumenty** na příkazovém řádku:
 - display *display* ... výběr X serveru
 - geometry *WxH{+-}X{+-}Y* ... velikost a poloha top-level okna
 - bg *color* ... barva pozadí
 - fg *color* ... barva textu a grafiky
 - fn *font* ... font pro text
 - name *name* ... jméno, pod kterým se budou číst resources
 - xrm *resourcestring* ... změna hodnot resources

- Neplést si server resources (okna, apod.) s resources ve smyslu konfiguračních souborů.
- Resource databáze X serveru je uložena v properties **RESOURCE_MANAGER** root okna obrazovky 0 a **SCREEN_RESOURCES** root oken jednotlivých obrazovek. Manipuluje se s ní pomocí programu **xrdb**.
- Nejvyšší prioritu má příkazový řádek, nejnižší prioritu mají hodnoty zakompilované v programu.
- Existují další standardní přepínače.
- Pomocí **-xrm** se dá nastavit cokoliv, co lze konfigurovat z resource souborů.

Slide 219

Pořadí čtení resources

1. Soubor *Classname* v adresáři `/usr/lib/X11/app-defaults`
2. Soubor *Classname* v adresářích zadaných proměnnými prostředí `XUSERFILESEARCHPATH` nebo `XAPPLRESDIR`
3. Properties nastavené pomocí `xrdb`, pokud nejsou, tak soubor `.Xdefaults` v domovském adresáři
4. Soubor zadaný proměnnou prostředí `XENVIRONMENT`, když není nastavena, tak soubor `.Xdefaults-hostname` v domovském adresáři
5. Argumenty z příkazové řádky

- Všechny soubory a properties se poskládají do jediné databáze.
- Přednost mají později načtené položky.
- *Classname* označuje třídu aplikace, blíže viz slidy popisující formát resources.

Slide 220

Resource soubory a properties

- Řádky ve tvaru *jméno : hodnota*
- Komentářové řádky začínají '! '.
- Vkládání souborů pomocí `#include "filename"`.
- Program `xrdb` ukládá resources ze souboru do property `RESOURCE_MANAGER` root okna obrazovky 0 a do `SCREEN_RESOURCES` root oken jednotlivých obrazovek.
- Navíc `xrdb` spouští na načítané soubory C preprocesor.
- `XResourceManagerString()` ... vrací obsah `RESOURCE_MANAGER`
- `XScreenResourceString()` ... vrací obsah `SCREEN_RESOURCES`

- Direktivu `#include` v resource souborech podporuje přímo `Xlib`.

Slide 221

Funkce pro manipulaci s resources

- `XrmInitialize()` ... inicializace databáze
- `XrmParseCommand()` ... čte argumenty z příkazové řádky
- `XrmGetFileDatabase()` ... čte resources ze souboru
- `XrmGetStringDatabase()` ... čte resources z řetězce
- `XrmMergeDatabases()` ... spojí databáze vytvořené předchozími funkcemi do jedné, ve které se pak budou vyhledávat hodnoty
- `XrmGetResource()` ... vrátí hodnotu z databáze
- `XrmEnumerateDatabase()` ... volá funkci pro všechny položky v databázi

- Program musí volat `XrmInitialize()` před použitím ostatních funkcí pracujících s resource databází.
- Funkce `XrmParseCommand()` není limitovaná na standardní X-ové přepínače. Je to náhrada za `getopt()`. Zpracované argumenty vyhodí z `argc`, `argv` a ukládá je tak, aby je mohly používat ostatní funkce pracující s resources.
- `XrmGetStringDatabase()` se dá použít pro čtení obsahu `RESOURCE_MANAGER` nebo hodnoty přepínače `-xrm`.

Slide 222

Řádek databáze resources

jméno : hodnota

object...subobject...resourcename: value

- Jednotlivé komponenty jména jsou jména **tříd**
`Xmail.ButtonBox.CommandButton.BackgroundColor: red`
- nebo **instancí**
`xmail.toc.includeButton.backgroundColor: blue`
- Místo jména komponenty (kromě poslední) se dá použít wildcard
`xmail.?.?.Background: antique white`
- Komponenty se dají spojovat pomocí **těsné (tight, '.')** a **volné (loose, '*')** vazby (**binding**). Volná vazba je wildcard pro libovolný počet mezilehlých komponent.

- `Object` je typicky jméno klienta (buď jméno programu, nebo zadané pomocí přepínače `-name`).
- Jména instancí začínají malým, jména tříd velkým písmenem.
- Vazba může předcházet před první, ale nesmí následovat za poslední komponentou. Před první komponentou se implicitně předpokládá těsná vazba.
- Otazník zastupuje jednu komponentu. Hvězdička spojuje komponenty.

Slide 223

Řádek databáze resources (2)

- Hodnota může obsahovat libovolné znaky.
- Do hodnoty nepatří úvodní mezery a tabulátory (pokud před nimi není backslash).
- Hodnota může pokračovat na dalším řádku, pokud je na konci řádku backslash.
- Fungují escape sekvence:
 - `\space` ... mezera (používá se na začátku hodnoty)
 - `\tab` ... tabulátor (na začátku hodnoty)
 - `\n` ... newline
 - `\\` ... backslash
 - `\ooo` ... oktalový zápis znaku

- Mezery před a za jménem (před dvojtečkou) se také ignorují.

Slide 224

Hledání v databázi

- `XrmGetResource()` ... zadávají se vždy dva řetězce (bez otazníků a hvězdiček), jeden se jmény tříd, druhý se jmény instancí. Algoritmus hledání najde nejlépe odpovídající položku v databázi.
- V databázi nejsou dvě položky se stejným jménem (nechá se pouze poslední načtená).
- Pokud víc položek odpovídá zadanému klíči, porovnávají se po komponentách zleva doprava a vezme ta s nejvyšší prioritou, pokračuje se, až zůstane jediná položka.
- Priorita je dána třemi pravidly, první pravidlo, které uspěje, definuje prioritu mezi položkami.

- Není úplně jasné, jak je to s postupem vyhodnocení: manpage říká zleva doprava, Xlib Programming Manual (O'Reilly) zprava doleva.

Pravidlo 1

Položka, která obsahuje aktuálně testovanou komponentu (ať už jako jméno, třídu, nebo '?') má přednost před položkou, která komponentu přeskakuje pomocí '*'.

Slide 225

| | |
|--|------------------|
| <code>*topLevel.quit.background:</code> | a |
| <code>*topLevel.Command.background:</code> | a |
| <code>*topLevel.?.background:</code> | má přednost před |
| <code>*topLevel*background:</code> | |

- Znak '*' nahrazuje libovolný počet komponent, ale '?' jen jednu.

Pravidlo 2

Položka, která obsahuje jméno aktuálně testované komponenty, má přednost před položkou, která obsahuje třídu nebo '?'. Položka s třídou má přednost před '?'.

Slide 226

```
*quit.background:      má přednost před  
*Command.background:   má přednost před  
*?.background:
```

- Čím specifitěji je komponenta zadána, tím má vyšší prioritu.
- Takto lze např. definovat barvu všech typů widgetů ('?'), předefinovat ji pro tlačítka (třída) a ještě jednou předefinovat pro jedno určité tlačítko (jméno).

Slide 227

Pravidlo 3

Položka, před kterou je těsná vazba, má přednost před položkou s volnou vazbou.

*box.background: má přednost před

*box*background:

Příklad kombinace pravidel 2 a 3:

*box*background: má přednost před

*box.Background:

protože pravidlo 2 má vyšší prioritu.

- V rozumně nadefinované resource databázi jsou obvykle priority vidět intuitivně, bez nutnosti ověřovat jednotlivá pravidla.

Slide 228

Podpora Xlib pro window manager

- **Substructure redirection** ... umožňuje řídit rozložení top-level oken
- **Reparenting** ... používá se pro přidávání rámečků oken
- **Save-set** ... zajišťuje obnovení oken při neočekávaném ukončení window manageru
- Prostředky pro standardní komunikaci ostatních klientů s window managerem.
- V jednu chvíli může být na jedné obrazovce aktivní pouze jeden window manager.

- Konfigurace X je obvykle taková, že ukončení window manageru znamená konec uživatelské relace.
- Komunikace s window managerem je popsána v ICCCM.
- Xlib sice poskytuje speciální funkce jako `XSetWMNormalHints()`, `XSetWMProtocols()` (pro klienty) a `XGetWMNormalHints()`, `XGetWMProtocols()` (pro window manager), ale tato komunikace probíhá pomocí mechanismů obecně používaných pro komunikaci mezi klienty (properties, posílání zpráv).

Slide 229

Substructure redirection

- Window manager nastaví masku událostí `SubstructureRedirectMask` na root okno.
- Pokus jiného klienta o změnu konfigurace top-level okna se neprovede. Místo toho se window manageru pošle událost `CirculateRequest`, `ConfigureRequest` nebo `MapRequest`. V události je popis požadované změny.
- Window manager má tři možnosti:
 1. Provést požadavek klienta nezměněný.
 2. Provést požadavek s upravenými argumenty.
 3. Zamítnout požadavek.
- Atribut okna `override_redirect` znamená, že se pro toto okno neuplatňuje substructure redirection.

- Konfigurace okna je pozice, velikost, šířka okraje a stacking order, namapování.
- Window manager provede změnu tak, že zavolá stejnou funkci, jakou volal klient. Protože má nastavenou masku událostí `SubstructureRedirectMask`, X server ví, že nemá znovu vygenerovat událost `*Request`, ale funkce se má skutečně provést.

Reparenting

- Při žádosti o klienta o namapování top-level okna dostane window manager (díky substructure redirection) událost `MapRequest`.
- Window manager vytvoří rámeček – nové, o trochu větší top-level okno.
- Původní top-level okno vloží do rámečku funkcí `XReparentWindow()`.
- Namapuje rámeček i top-level okno klienta uvnitř.
- Když window manager posune top-level okno, X server neregeneruje `ConfigureNotify` pro klienta, protože se nemění relativní poloha top-level okna vůči rámečku. Událost `ConfigureNotify` pošle klientovi window manager.

Slide 230

- Window manager může využít rámeček, který je součástí okna, obvykle to ale nedělá.
- Parametry syntetické události `ConfigureNotify` vygenerované window managerem obsahují souřadnice relativně vůči root oknu.

Save-set

- Save-set je seznam oken ostatních klientů, která jsou spravována window managerem.
- Window manager přidává top-level okna klientů do save-set pomocí funkce `XAddToSaveSet()`.
- Window manager provádí reparenting (kvůli přidávání rámečků) a odmapování oken (když místo nich zobrazuje ikony). Save-set zajišťuje, že při ukončení window manageru se okna znovu namapují a obnoví se jejich původní vlastník.

Slide 231

- Každý klient může používat svůj save-set, ale typicky to dělají pouze window managery – ostatní klienti žádná cizí okna nespravují.

Slide 232

Konec