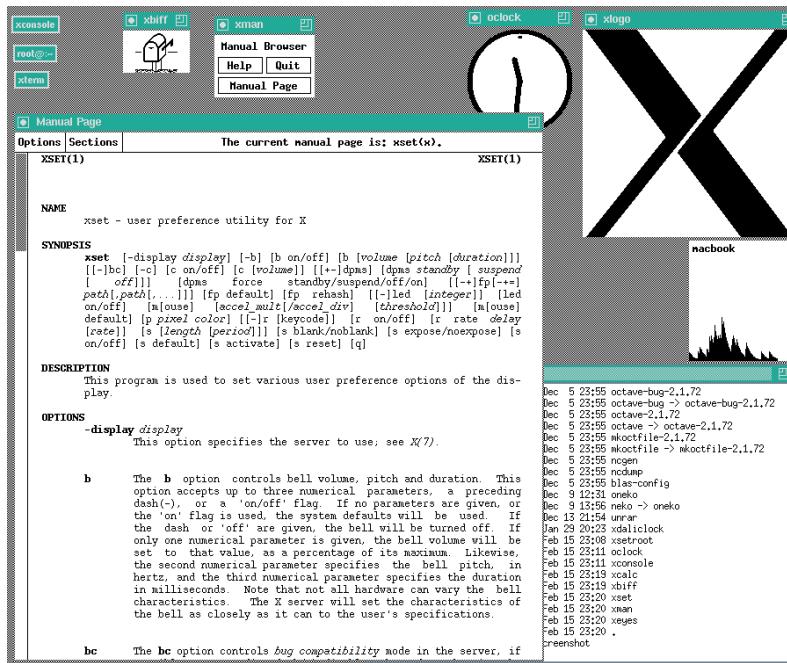


واسطه گرافیکی کاربر

لینوکس به عنوان یک سیستم عامل میزکار یا همان دسکتاپ، نیازمند واسطه گرافیکی کاربر یا GUI می‌باشد. یادگیری و استفاده از GUI لینوکس بسیار ساده و راحت است. با وجود شباهت‌هایی که بین محیط گرافیکی دسکتاپ لینوکس با ویندوز و مکینتاش وجود دارد، یک تفاوت اساسی نیز در این میان مشهود است و آن حق انتخاب_ یا انتخاب_ هایی است که شما درباره قسمت‌های گرافیکی و چگونگی ترکیب آنها با یکدیگر دارید.

GUI های لینوکس از ۳ قسمت تشکیل شده‌اند:

- اولین بخش آن Window System X است، که یک سیستم نرم‌افزاری و یک پروتکل شبکه می‌باشد که وظیفه آن تأمین یک واسطه گرافیکی کاربر(GUI) می‌باشد.
- دومین قسمت Window Manager است که بر روی X قرار می‌گیرد. این قسمت در واقع این امکان را به شما می‌دهد که بیش از یک برنامه کاربردی GUI را اجرا کنید. بدون این بخش شما توانایی جابه‌جایی پنجره‌ها، تغییر اندازه آنها و کارهای مشابه دیگر را نخواهید داشت.
- سومین قسمت محیط میزکار یا Desktop Environment نامیده می‌شود. این قسمت نیز مانند Manager بر روی X قرار می‌گیرد و ویژگی‌هایی دارد که به لینوکس امکان مجتمع سازی کاربرد ها را می‌دهد. (همانند COM و OLE در دنیای مایکروسافت). علاوه بر این Desktop Environment توانایی های Window Manager را توسعه می‌دهد به طوریکه بتواند بیشتر امکانات و کاربردهای GUI را به کاربران لینوکس ارائه دهد. که از آن جمله می‌توان به محیط‌های GNOME و KDE اشاره کرد.



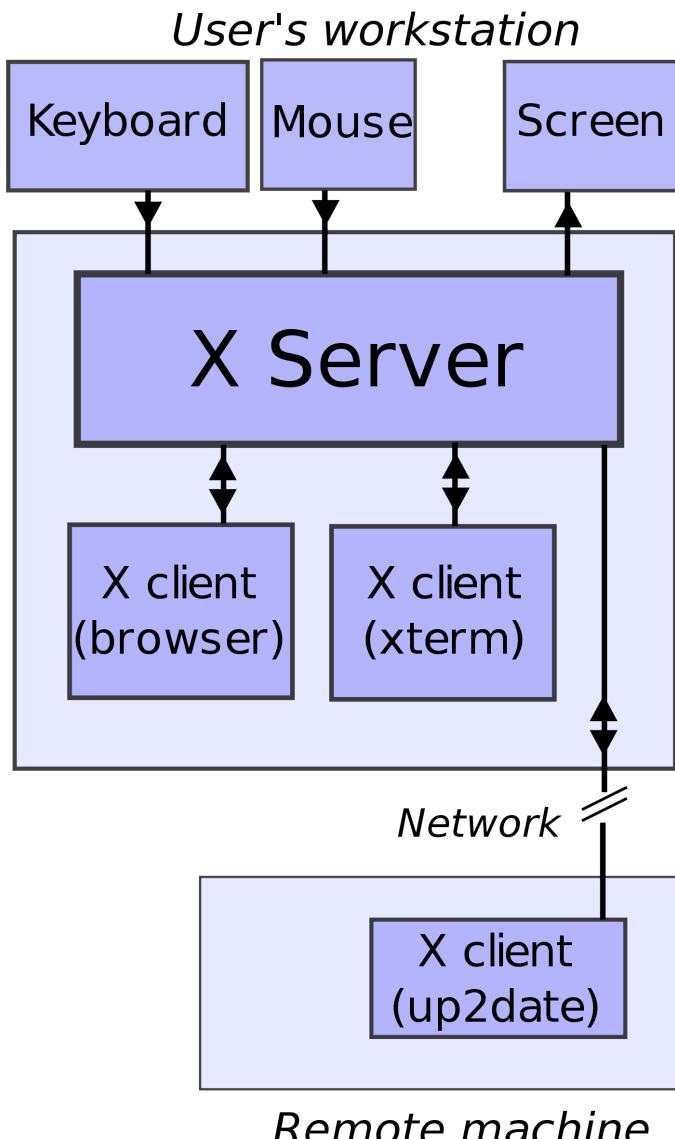
تاریخچه X

X در سال ۱۹۸۴ در دانشگاه MIT به وجود آمد. نسخه کنونی پروتکل X یعنی X11 در سپتامبر ۱۹۸۷ ظاهر شد. بنیاد X.Org این پروژه را براساس مرجع پیاده سازی شده کنونی هدایت می‌کند و سرویس دهنده X.Org به عنوان نرم افزار آزاد و منبع باز تحت مجوز MIT در دسترس می‌باشد.

اصول اولیه

در ۱۹۸۴، باب شیفلر و جیم گتیس اصول اولیه ایکس را به این صورت قدوین کردند:

- هیچ قابلیت جدیدی اضافه نکنید مگر اینکه برنامه‌نویسی بدون آن قابلیت نتواند برنامه واقعی‌اش را تمام کند.
 - اهمیت تصمیم‌گیری در این مورد که برنامه چه چیزی قرار نیست باشد، همانقدر مهم است که برنامه چه چیزی قرار است باشد. لازم نیست به همه نیازهای جهان پاسخ دهید. نکته مهم این است که برنامه قابل توسعه باشد تا بعدها بتوان با حفظ سازگاری، آن را پیش برد.
 - تنها چیزی که از تعمیم دادن بر مبنای یک مثال بدتر است، تعمیم دادن بدون وجود حتی یک مثال است.
 - اگر مشکل کاملاً درک نشده باشد، بهترین کار این است که هیچ جوابی داده نشود.
 - اگر می‌توانید ۹۰٪ خواسته‌ها را با ۱۰ درصد کار جواب دهید، از راه حل ساده‌تر استفاده کنید.
 - تا جایی که ممکن است، پیچیدگی را ایزوله کنید.
 - به جای خط مشی، مکانیزم ارائه دهید. به طور عام، خط مشی رابط کاربری را به کلاینت‌ها بسپارید.
- در طول توسعه X11، قانون اول به این شکل تغییر کرد: «هیچ قابلیت جدیدی اضافه نکنید مگر اینکه برنامه‌ای واقعی را بشناسید که به آن نیاز داشته باشد.»



مفاهیم اساسی

همانطور که پیش از این گفته شد، X تنها یکی از سه قسمت نرم افزاریست که لینوکس برای داشتن GUI بر روی سیستم به آن احتیاج دارد. اما در اصل یک بخش اساسی محسوب می‌شود. در واقع X تنها جعبه ابزاریست که ابزار گرافیکی اصلی را شامل می‌شود. به عنوان مثال اگر فردی بخواهد یک برنامه پردازشگر متن بنویسد، باید بداند چگونه متن را روی صفحه نمایش دهد و فونت‌ها و اندازه متن به چه ترتیبی باشد. و یا شاید نیاز به کشیدن خطوطی برای ایجاد جدول باشد. و این جاست که X به درد می‌خورد، X دارای API‌هایی است که قادرند متن را با فونت و فرمان‌های داده شده برای رسم خطوط نمایش دهند.

خود X از سه قسمت اصلی تشکیل شده است:

- سرور X
- مجموعه‌ای از کتابخانه‌های گرافیکی
- مجموعه‌ای از برنامه‌های کاربردی گرافیکی که معمولاً از کتابخانه‌های گرافیکی استفاده می‌کنند.

سرور X چیست؟

در لینوکس و سایر سیستم عامل های شبه یونیکس (Unix-like) برای نمایش گرافیک از یک سیستم سرویس دهنده - سرویس گیرنده یا client-server استفاده می شود. سرور X برنامه ای است که اطلاعات را از صفحه کلید و ماوس کاربر دریافت می کند و در مانیتور نمایش می دهد. سرویس گیرنده یا client نیز برنامه ای است که فعالیت های کاربر را به سرور X می فرستد، مانند تکان دادن ماوس.

در واقع وقتی شما یک لینوکس با محیط گرافیکی نصب می کنید همزمان یک شبکه داخلی نصب کرده اید. توجه داشته باشید که سرور X می تواند در کامپیوتر دیگری در یک شبکه دیگر نصب شود و شما از طریق آدرس دهی به آن سرور دسترسی داشته باشید . در حقیقت این روش Client-Server بودن نمایش در یونیکس این مزیت را به همراه دارد که شما می توانید سرور X را در جایی دیگر بر روی سیستم عاملی دیگر اجرا کنید و از طریق Xclient به آن سیستم وصل شوید ، مانند این که پشت همان کامپیوتر راه دور نشسته اید.

انواع سرور X

همان طور که در بخش های قبل توضیح داده شد X یک سیستم نرم افزاری و یک پروتکل شبکه می باشد بنابراین هر شخص یا شرکتی می تواند برنامه ای بنویسد که نقش Xserver را ایفا کند. انواع مختلف X سرور وجود دارد که بعضی تجاری هستند بعضی غیر تجاری ، از انواع غیر تجاری می توان Xfree86 و X.Org را نام برد و از انواع تجاری می توان به محصولات دو شرکت Metro Link و Xi Graphics اشاره کرد . در حال حاضر در اغلب توزیع های گنو/لینوکس از X.Org استفاده می شود .

اجرای X

برای اجرای x کافیست بعد از نصب و تنظیم ابتدا در حالت متني وارد سیستم شوید(login) و سپس در مقابل اعلان فرمان بنویسید startx با این کار سرور X در کامپیوتر شما اجرا خواهد شد.

مفهوم مدیر پنجره یا Window Manager

بعد از اجرای X server شما می توانید بدون وارد شدن به مدیر پنجره به راحتی اغلب برنامه ها را اجرا کنید ، اما خواهید دید که هیچ امکانی برای بزرگ و کوچک کردن پنجره ها وجود ندارد ، از طرفی شما نمی توانید چند پنجره را هم زمان مدیریت کنید ، بنابر این به یک مدیر پنجره یا Window Manager احتیاج خواهید داشت. وظیفه ی مدیر پنجره رسم خطوط اطراف پنجره هاست و فراهم آوردن قابلیت تغییر اندازه پنجره ها ، وظیفه ی دیگر یک مدیر پنجره انتخاب پنجره های فعال است ، پنجره های که شما در آن کار می کنید و اغلب رنگ قسمت بالای آن (title bar) پرنگ تر از سایر پنجره هاست را پنجره ی focus شده می گویند. در حال حاضر در سیستم عامل های مختلف انواع زیادی focus وجود دارد از جمله:

۱. click-to-focus

پنجره ای انتخاب می شود که بر روی آن کلیک شده است.

۲. focus-follow-mouse

پنجره ای انتخاب می شود که ماوس ببروی آن قرار دارد و نیازی به کلیک نیست ، هر جا که ماوس شما باشد آن پنجره فعال می شود و titlebar آن پر رنگ تر از سایرین نمایش داده می شود.

بسیاری از مدیر پنجره ها منوهایی برای تنظیم مدیر پنجره و پنجره ها در اختیار کابران قرار می دهند که اغلب با یک کلیک راست بر روی میز کار قابل دسترسی است. از جمله امکاناتی که اغلب مدیر پنجره ها در لینوکس فراهم آورده اند امکان pager است . یک برنامه است که به شما این امکان را می دهد که بتوانید از بیش از یک محیط کار (workspace) در صفحه نمایش استفاده کند ، برای نمونه در یکی لغت نامه و در دیگری یک برنامه تایپ را اجرا کنید و با این تقسیم کار از شلوغ شدن محیط کارتان جلوگیری کنید.

باید توجه داشته باشید که یک مدیر پنجره نمی تواند محتوای پنجره ها را مدیریت کند. منظور از محتوا دکمه ها و

منوهای خود برنامه است مانند منوی file یا دکمه OK که در اغلب برنامه ها جود دارد.

چند نمونه مدیر پنجره

در این بخش به چند نمونه از مدیر پنجره در لینوکس اشاره خواهیم کرد.

KWin : این مدیر پنجره بخشی از محیط میز کار KDE می باشد که وظیفه ای مدیریت پنجره در این محیط را دارد.

Metacity : این مدیر پنجره اغلب در محیط میز کار GNOME استفاده می شود و در واقع مدیر پنجره ای پیش فرض گنوم به شمار می آید. برای اطلاعات بیشتر درباره ای این مدیر پنجره می توانید به آدرس مقابل مراجعه کنید:

<http://www.gnome.org/softwaremap/projects/metacity>

Xfwm : این مدیر پنجره در محیط میز کار Xfce استفاده می شود.

Sawfish : این مدیر پنجره در محیط میز کار GNOME نسخه های 1.2 تا 1.4 پیدا می شود و در حال حاضر از آن کمتر استفاده می شود. برای اطلاع از این میز کار نیز می توانید به این ادرس مراجعه کنید:
<http://sawmill.sourceforge.net>

IceWM : این مدیر پنجره window manager کم آن قابلیت های بسیار زیادی در اختیار استفاده کنندگان قرار می دهد مانند pager (پیجو!) یا مدیریت آسان پنجره ها . برای کسب اطلاعات بیشتر به سایت رسمی آن در www.icewm.org مراجعه فرمایید.

WindowMaker : این مدیر از واسط قدیمی NeXT الهام گرفته و ظاهری شبیه به آن دارد ، هدف این مدیر پنجره فراهم آوردن محیطی است با حجم کم مانند IceWM که به حافظه کمتری نیاز داشته باشد. این مدیر پنجره می تواند به عنوان مدیر پنجره ای محیط میز کار KDE قرار بگیرد. (به جای KWM) برای اطلاعات بیشتر به صفحه ای اصلی آن در www.windowmaker.org مراجعه کنید.

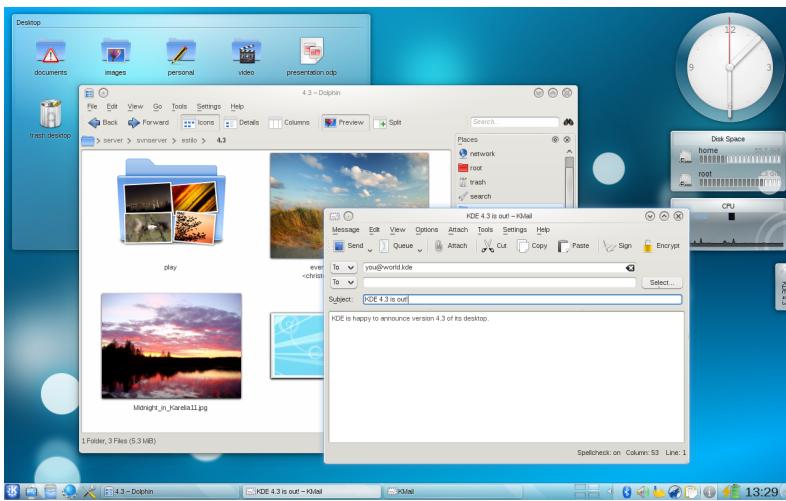
محیط میز کار یا Desktop Environment چیست؟

Desktop Environment یا ترجمه ای فارسی آن که «محیط میز کار» گفته می شود و اغلب با مخفف DE مشخص می شود همان طور که Window Mnager با WM نمایش داده می شود.

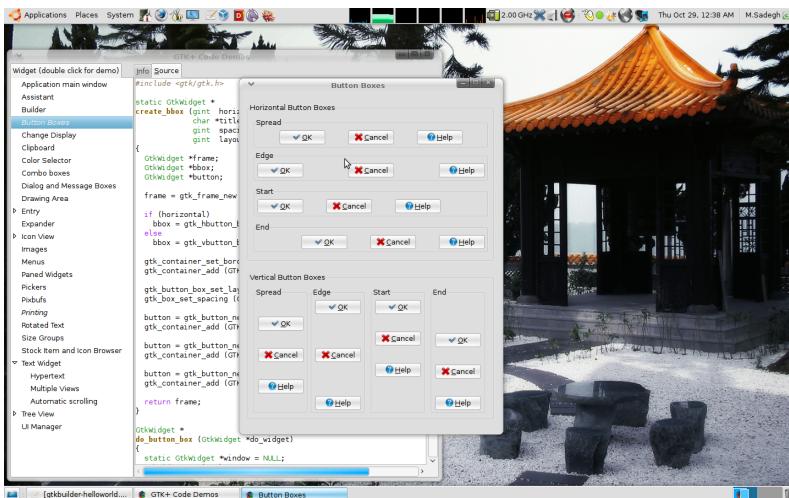
یک مدیر پنجره با این که ابزاریست مفید اما نمی تواند یک واسط کابر گرافیکی (GUI) مناسب را تدوین کند. کابران OS/2 یا Mac OS با محیط GUI کار کرده اند و می دانند در چنین محیطی ابزارهای بسیاری در دست دارند تا با آنها کار کنند. از ابزار تنظیم گرفته تا ابزاری برای نمایش ساعت یا ماشین حساب یا ویرایشگر متن یکی از ویژگی های جالب این محیط ها تنظیم یک فونت خاص برای هر برنامه است .

یک محیط میز کار تمام این مسائل را حل می کند و میز کاری مشابه و حتی بهتر از windows در اختیار شما قرار می دهد ، در واقع Desktop Environment ها یک واسط گرافیکی کامل در اختیار شما قرار می دهند.

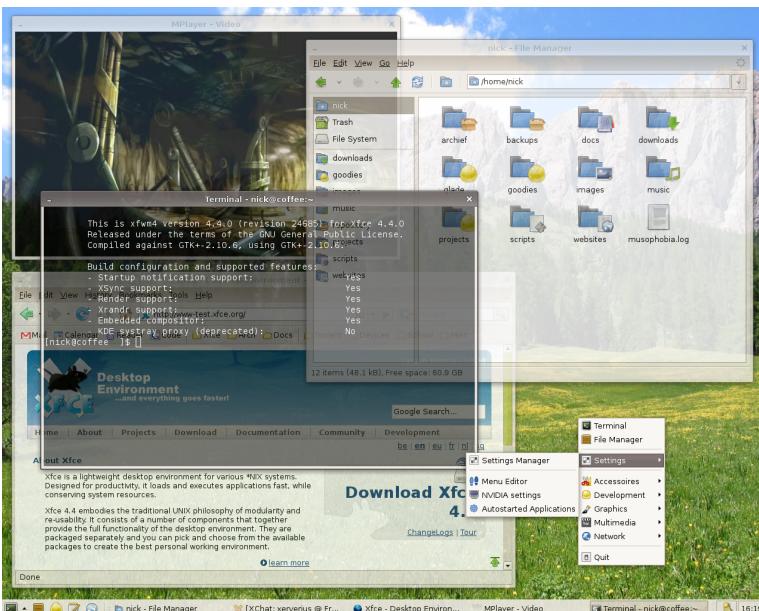
محیط های میز کار مشهور در گنو/لینوکس



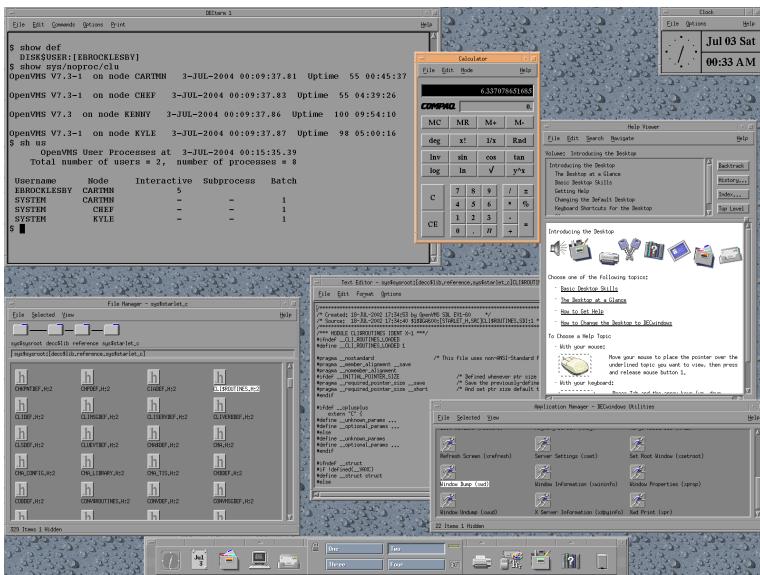
(تلفظ کنید : کی دی ای) این میز کار که مخفف K Desktop Environment می باشد یکی از محبوب ترین میز کارهای لینوکس به شمار می آید که از مدیر پنجره‌ی پیش فرض KWin استفاده می کند و از Widgetset های Qt برای نمایش پنجره ها و محتوای آنها بهره می برد. برای کسب اطلاعات بیشتر درباره ای این میز کار به سایت رسمی آن یعنی www.kde.org مراجعه کنید.



(تلفظ کنید: گنوم نوم) این میز کار نیز که مخفف GNU Network Object Model است از دیگر محیط های میز کار محبوب لینوکس کارهاست که مانند KDE در توزیع های فراوانی به صورت پیش فرض تعییه شده است. و از مدل پنجره‌ی Metacity به همراه GTK+ های widgetset می کند. برای کسب اطلاعات بیشتر به سایت رسمی آن یعنی www.gnome.org مراجعه کنید.



(تلفظ کنید: ایکس اف سی ئی) این میز کار که یک میز کار خلاصه و جمع و جور است، با حجم کم و نیازمندی های سخت افزاری پایین تر نسبت به GNOME و KDE یکی از میز کارهای مشهور لینوکس است. این میز کار از Widgetset های + GTK استفاده می کند. برای اطلاعات بیشتر به سایت مادر در www.xfce.org مراجعه فرمایید.



تا کنون در باره‌ی محیط‌های میز کار بازمنت سخن به میان آورده‌یم اما محیط Commerical یک محیط تجاری یا CDE است که در یونیکس‌های تجاری استفاده می‌شود. CDE مخفف Common Desktop Environment است و از Motif‌های widgetset برای اطلاعات پیشتر به سایت www.xig.com مراجعه فرمایید.

اجرای یک محیط میز کار

برای اجرای این محیط‌های میز کار اگر از KDM و GDM استفاده می‌کنید با اجرای X به طور خودکار یکی از این دو اجرا شده و از شما می‌پرسند که به کدام میز کار هدایت‌تان کنند، اما اگر از GDM و KDM استفاده نمی‌کنید، بعد از ورود از طریق خط فرمان و دیدن اعلان خط فرمان می‌توانید برای دسترسی به KDE startkde تایپ کنید.

بعد از اجرای این دستورات محیط‌های میز کارشما بالا آمده و شما می‌توانید از آن‌ها استفاده کنید. برای شروع می‌توانید از منوی آن‌ها کمک بگیرید، به منوی KDE که در سمت چپ و پایین قرار دارد K menu می‌گویند که اغلب یک K بزرگ بروی آن است و به منوی GNOME که اغلب یک جای پا که نشانه‌ی گنوم است بروی آن دیده‌ی شود G menu گفته می‌شود. برای کار با فایل‌ها نیز برنامه‌ای در هر محیط میز کار تعییه شده که بهFileManager یا مدیر فایل مشهور است مانند Dolphin در KDE و Natilus در GNOME البته مدیران فایل دیگری نیز وجود دارند و این مدیران فایل یا فایل منیجرها را می‌توان در محیط‌های دیگر نیز استفاده کرد برای نمونه از ناتیلوس در KDE استفاده کرد.

Widget Set مفهوم

تا این جا درباره‌ی مفاهیم پایه‌ای سخن گفتیم مفاهیمی که توسط آن‌ها دانستید که چگونه پنجره‌ها و محیط گرافیکی بر روی صفحه نمایش شما دیده می‌شوند، اکنون به بخش دیگری از این مطالب می‌پردازیم، یکی از مهم ترین اجزای هر میزکاری پنجره‌ها، منوها و جعبه‌های گفتگو هستند، مدیریت این بخش‌ها را WidgetSet یا به عهده دارند.

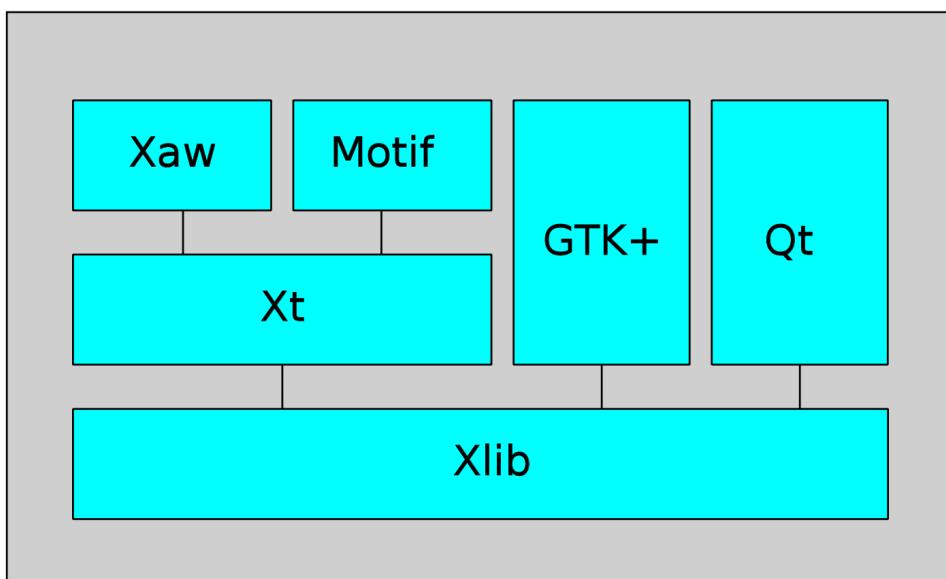
یکی از مهم ترین مفاهیم که روش نمایش و رفتار منوها و جعبه‌های گفتگو را در X سرور تعریف می‌کند ابزارهای برنامه نویسی به نام Widget set هستند، Widget set ها در واقع محتویات درون پنجره را کنترل می‌کنند. گاهی در برنامه‌ای برای انتخاب گزینه‌ها از منو باید دکمه‌ی ماوس خود را پایین نگاه دارید اما در برخی برنامه‌ها نیازی به این کار نیست و با حرکت ماوس بر روی گزینه‌های منوها گزینه‌ها انتخاب می‌شوند، این همان widget set است که این رفتارها را تعریف می‌کند. البته در سیستم عامل گنو/لینوکس هر برنامه‌ای می‌تواند از widget set های خود استفاده کند، برای نمونه شما در محیط میزکار GNOME برنامه‌ی gaim و xpdf را با یکدیگر مقایسه کنید، هر کدام به نحوی منوها و پنجره‌ها را مدیریت می‌کنند این به این دلیل است که هر کدام از یک سری widget استفاده می‌کنند.

در حال حاضر دو سری از widget set ها هستند که بسیار محبوبند، یکی widget set های Qt و دیگری GTK+ های set است. گفتن این نکته نیز لازم است که در محیط‌های قدرتمندی چون KDE یا GNOME شما می‌توانید نوع widget set های مورد استفاده را تغییر بدهید. شاید شنیده باشید که می‌گویند «برنامه‌ای برای KDE

نوشته شده است» یا «برنامه ای برای GNOME نوشته شده است» مفهوم این جملات آن است که برنامه‌ی اول از Qt و برنامه‌ی دوم از GTK+ استفاده می‌کند، خود GTK+ در واقع «کتابخانه»‌هایی هستند که در آن‌ها برنامه‌هایی برای کنترل Widget‌ها نوشته شده است، اکنون این کتابخانه‌ها یا library‌ها به چه کاری می‌آیند؟ این کتابخانه‌ها باعث می‌شوند تا برنامه نویس به جای آن که از ابتدا وقت خود را برای نوشتن این رفتار‌ها (رفتارهایی که widget‌ها تعریف می‌کنند) تلف کنند در برنامه‌ی خود یک لینک به این کتابخانه‌ها می‌دهند، از این پس برنامه‌ی نوشته شده با تعاریفی که از قبل برای آن در کتابخانه‌ها شده است کار می‌کند، بنابراین یک برنامه‌ی KDE به کتابخانه‌های Qt لینک داده شده و از widget‌های تعریف شده در آن استفاده می‌کند، این همان دلیل شbahat برنامه‌های KDE به هم است، این قضیه درباره‌ی گنوم هم صادق است. شاید این سوال برای شما پیش بیاید که آیا می‌شود از برنامه‌های GNOME در KDE استفاده کرد؟ (یا بلعکس)، در پاسخ باید گفت بله، در اغلب موارد می‌توان استفاده کرد اما برای استفاده از هر کدام باید کتابخانه‌هایشان را نیز نصب کنید برای نمونه برای استفاده از برنامه‌های KDE باید حتماً کتابخانه‌های Qt را نصب کرده باشید.

منابع :

<http://www.wikipedia.org> , <http://www.gniran.org> , <http://www.govashir.com>



برنامه نویسی GUI

در محیط X از کتابخانه Xlib می‌توان برای برنامه نویسی GUI بهره برد. Xlib یک کتابخانه سرویس گیرنده پروتکل X Window System می‌باشد که به زبان C نوشته شده است. این کتابخانه حاوی توابعی برای فعل و انفعال با یک سرور X می‌باشد. این توابع به برنامه نویس اجازه می‌دهد بدون آنکه با جزئیات این پروتکل آشنایی داشته باشد برنامه بنویسد. برنامه‌های کاربردی اندکی به طور مستقیم از Xlib استفاده می‌کنند. بلکه کتابخانه‌های دیگری وجود دارند که از Xlib برای فراهم نمودن widget toolkits استفاده می‌کنند. که از آن جمله می‌توان به موارد ذیل اشاره کرد:

- Intronics(Xt)
- Athena widget set(Xaw)
- Motif
- Gtk+
- Qt (X11 version)
- Tk

Xlib در سال ۱۹۸۵ ظاهر شد و هم اکنون در واسط گرافیکی کاربر تعداد زیادی از سیستم‌های عامل شبیه یونیکس به کار برده می‌شود. کتابخانه XCB تلاشی است برای جایگزینی Xlib. با این حال هنوز هم Xlib به طور وسیعی

استفاده می شود ، امروزه برنامه هایی که با استفاده از این کتابخانه پیاده سازی شده اند می توانند از XCB به عنوان یک لایه انتقال سطح پایین تر استفاده نمایند. مزایای XCB نسبت به Xlib یکی کاهش در اندازه و پیچیدگی کتابخانه و دیگری دسترسی مستقیم به پروتکل X11 می باشد.

برنامه ذیل با استفاده از کتابخانه Xlib یک پنجره ایجاد می نماید که در داخل آن یک مریع سیاه و یک نوشه وجود دارد، برای کامپایل این برنامه دستورات ذیل را در خط فرمان تایپ کنید:

```
$ gcc -Wall x-helloworld.c -o x-helloworld -lX11
```

```
-----  
/*  
Simple Xlib application drawing a box in a window.  
*/  
  
#include <X11/Xlib.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
int main(void) {  
    Display *d;  
    Window w;  
    XEvent e;  
    char *msg = "Hello, World!";  
    int s;  
  
    /* open connection with the server */  
    d = XOpenDisplay(NULL);  
    if (d == NULL) {  
        fprintf(stderr, "Cannot open display\n");  
        exit(1);  
    }  
  
    s = DefaultScreen(d);  
  
    /* create window */  
    w = XCreateSimpleWindow(d, RootWindow(d, s), 10, 10, 200, 200, 1,  
                           BlackPixel(d, s), WhitePixel(d, s));  
  
    /* select kind of events we are interested in */  
    XSelectInput(d, w, ExposureMask | KeyPressMask);  
  
    /* map (show) the window */  
    XMapWindow(d, w);  
  
    /* event loop */  
    while (1) {  
        XNextEvent(d, &e);  
        /* draw or redraw the window */  
        if (e.type == Expose) {  
            XFillRectangle(d, w, DefaultGC(d, s), 20, 20, 10, 10);  
            XDrawString(d, w, DefaultGC(d, s), 50, 50, msg, strlen(msg));  
        }  
        /* exit on key press */  
        if (e.type == KeyPress)  
            break;  
    }  
  
    /* close connection to server */  
    XCloseDisplay(d);  
  
    return 0;  
}
```

همانطور که ملاحظه نمودید ایجاد واسط گرافیکی کاربر با استفاده مستقیم از Xlib تا حدی پیچیده می‌باشد بنابراین برنامه‌نویسان به فکر ایجاد API ها سطح بالایی افتادند که تاحد زیادی کار با آن‌ها ساده باشد. صد البته توسعه دهنده‌گان نرم افزارهای آزاد و متن باز نیز از پیشگامان این حرکت بودند که از آن جمله می‌توان به کتابخانه پرقدرت و محبوب GTK+ اشاره کرد.

GIMP toolkit (Gimp) یک مجموعه ویجت چند بستری برای ایجاد واسط گرافیکی کاربر می‌باشد که به همراه Qt یکی از جعبه ابزارهای محبوب سیستم پنجره‌ای X می‌باشد.

GNU Image Manipulation Program (GIMP) در اصل برای توسعه نرم افزار GIMP (GNU Image Manipulation Program) که یک ویراشگر تصاویر می‌باشد در سال ۱۹۹۷ توسط Peter Mattis و Spencer Kimball که از اعضای XCF بودند در دانشگاه برکلی ایجاد شد.

GIMP بخشی از پروژه گنو بوده و تحت مجوز LGPL انتشار یافته است بنابراین با استفاده از آن می‌توانید نرم افزار باز، نرم افزار آزاد یا هر نوع نرم افزار تجاری غیرآزاد را ایجاد نمایید بدون اینکه مبلغی جهت کسب مجوز یا حق امتیاز آن بپردازید.

تاریخچه

GTK در اصل طراحی شده بود تا جایگزینی برای کتابخانه Motif که در توسعه GIMP استفاده می‌شد باشد. Petter Mattis هنگامی که بر روی پروژه GIMP کار می‌کرد گاهی اوقات از Motif دلسربد می‌شد بنابراین تصمیم به نوشتن یک جعبه ابزار GUI گرفت و موفق شد در انتشار ۰.۶۰ گیمپ آن را به طور کامل با Motif جایگزین نماید. همچنین به علت گسترش GTK تصمیم گرفتند تا آن را بر اساس فناوری شی گرا دوباره نویسی نمایند که این کار هم با موفقیت انجام گرفت و نام GTK+ تغییر یافت که این نسخه جدید از کتابخانه اولین بار در انتشار ۰.۹۹ گیمپ به کار گرفته شد.

بعد از ۱ GTK+ ارائه شد. مشخصات جدیدی که در این نسخه تعییه شده بود شامل یک پردازشگر متن بهبود یافته با استفاده از کتابخانه Pango، یک Theme Engine جدید، دسترسی پذیری بهبود یافته با استفاده از کتابخانه ATK (محفظ Accessibility Toolkit)، انتقال کامل کتابخانه به یونیکد با استفاده از رشته‌های متñی UTF-8 و یک API بسیار انعطاف پذیر، با این حال GTK+2 فاقد سازگاری با GTK+1 می‌باشد و برنامه نویسان بایستی برنامه‌های خود را به آن انتقال دهند. با شروع نسخه ۲.۵.۲ GTK+2 به کتابخانه Cairo که برای پردازش گرافیک برداری طراحی شده بود وابسته شد.

ساختار

GTK+ یک مجموعه ویجت شی گرا می‌باشد که اگرچه به طور کامل به زبان C نوشته شده اما پیاده سازی آن با استفاده از نظریه کلاس‌ها و توابع callback (اشاره گرها به توابع) صورت پذیرفته است. شی گرایی این کتابخانه بوسیله کتابخانه Gobject (محفظ Glib Object System) محقق شده است.

GTK+ بر روی GDK (محفظ GIMP Drawing Kit) بنا شده که بطور اساسی این هم یک پوشش، پیرامون توابع سطح پایین، جهت دستیابی به توابع زیرین ایجاد کننده پنجره Xlib در سیستم مدیریت پنجره X می‌باشد.

در سرویس دهنده X11، این کتابخانه برای ترسیم ویجت‌ها از کتابخانه Xlib استفاده می‌نماید البته Xt می‌تواند به عنوان یک جایگزین پیشنهاد شود. استفاده از Xlib قابلیت انعطاف زیادی را برای GTK+ فراهم می‌کند و اجازه می‌دهد تا از GTK+ در پلتفرمی که X در دسترس نیست بتوان استفاده کرد. در حالیکه GTK+ به صورت پیش فرض X را هدف گیری نموده است ولی دیگر پلاتفرم‌ها نیز پشتیبانی می‌شوند مانند Direct Fb و Quartz و Mac روی Microsoft Windows و OSX.

همچنین جزء سومی به نام Glib وجود دارد که در برگیرنده تعداد اندکی جایگزین به جای برخی توابع استاندارد می‌باشد، به اضافه چند تابع اضافی برای اداره لیست‌های پیوندی و غیره. این توابع جایگزین به منظور افزایش قابلیت

حمل GTK+ مورد استفاده قرار گرفته ، به عنوان مثال برخی از توابعی که در اینجا پیاده گردیده‌اند بر روی یونیکس‌های دیگر قابل دسترسی نیستند و یا اینکه غیراستاندارد می‌باشند `(g_strerror)`. به علاوه بعضی از آنها در بردارنده افزایش کارایی برای نسخه‌های `libc` می‌باشند ، مانند `g_malloc` که دارای ابزارهای اشکال‌زدایی بهبودیافته می‌باشد. و در آخر ، GTK از کتابخانه Pango در جهت بین‌المللی سازی خروجی متن استفاده می‌کند .

GTK+ می‌تواند ظاهر ویجت را تغییر دهد ، این کار با استفاده از موتورهای نمایش مختلف انجام می‌شود. موتورهای نمایش مختلفی وجود دارند که تلاش می‌کنند ظاهر ویجت‌های پلتفرم در حال استفاده را تقلید نمایند.

Binding های زبان‌های برنامه نویسی کتابخانه‌ای که با استفاده از یک زبان برنامه نویسی نوشته شده است ممکن است در سایر زبان‌ها نیز مورد استفاده قرار بگیرد البته به شرط آن که آن زبان نوشته شده باشد. **GTK+** برای بسیاری از زبان‌های برنامه نویسی **binding** دارد.

پشتیبانی رسمی	binding	نام زبان
نسخه اصلی	GTK+	C
بله	Gtkmm (gtk--)	C++
بله	ruby-gtk2	Ruby
بله	PyGTK	Python
(نسخه ویندوزی وجود ندارد)	java-gnome	Java
بله	GTK#	C#
بله	PHP-GTK	PHP
خیر	Gtk2-Perl	Perl
خیر	GtkAda	Ada
خیر	gtkD	D
خیر	gtk2hs	Haskell
خیر		Lua
خیر	LablGTk	Ocaml
خیر		Pascal
خیر		Pike
خیر	seed	JavaScript
خیر	Smalltalk YX	Smalltalk
خیر		Euphoria
خیر		Other.NET
	زبانی که بر اساس سیستم Gobject ساخته شده است	Vala و GOB

شروع برنامه نویسی با GTK+

این برنامه دارای یک پنجره می‌باشد که عنوان آن «Hello World!» می‌باشد و در درون این پنجره یک برچسب با همان متن نمایش داده می‌شود، برای کامپایل این برنامه دستورات ذیل را در خط فرمان تایپ کنید:

```
$ gcc -Wall gtk-helloworld.c -o gtk-helloworld `pkg-config --cflags --libs gtk+-2.0`
```

```
-----  
helloworld.c
```

```
#include <gtk/gtk.h>

/* This is a callback function. The data arguments are ignored
 * in this example. More on callbacks below. */
static void hello( GtkWidget *widget,
                    gpointer   data )
{
    g_print ("Hello World\n");
}

static gboolean delete_event( GtkWidget *widget,
                             GdkEvent  *event,
                             gpointer   data )
{
    /* If you return FALSE in the "delete_event" signal handler,
     * GTK will emit the "destroy" signal. Returning TRUE means
     * you don't want the window to be destroyed.
     * This is useful for popping up 'are you sure you want to quit?'
     * type dialogs. */

    g_print ("delete event occurred\n");

    /* Change TRUE to FALSE and the main window will be destroyed with
     * a "delete_event". */

    return TRUE;
}

/* Another callback */
static void destroy( GtkWidget *widget,
                     gpointer   data )
{
    gtk_main_quit ();
}

int main( int   argc,
          char *argv[] )
{
    /* GtkWidget is the storage type for widgets */
    GtkWidget *window;
    GtkWidget *button;

    /* This is called in all GTK applications. Arguments are parsed
     * from the command line and are returned to the application. */
    gtk_init (&argc, &argv);

    /* create a new window */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* When the window is given the "delete_event" signal (this is given
     * by the window manager, usually by the "close" option, or on the
     * titlebar), we ask it to call the delete_event () function
     * as defined above. The data passed to the callback
```

```

/* function is NULL and is ignored in the callback function. */
g_signal_connect (G_OBJECT (window), "delete_event",
                  G_CALLBACK (delete_event), NULL);

/* Here we connect the "destroy" event to a signal handler.
 * This event occurs when we call gtk_widget_destroy() on the window,
 * or if we return FALSE in the "delete_event" callback. */
g_signal_connect (G_OBJECT (window), "destroy",
                  G_CALLBACK (destroy), NULL);

/* Sets the border width of the window. */
gtk_container_set_border_width (GTK_CONTAINER (window), 10);

/* Creates a new button with the label "Hello World". */
button = gtk_button_new_with_label ("Hello World");

/* When the button receives the "clicked" signal, it will call the
 * function hello() passing it NULL as its argument. The hello()
 * function is defined above. */
g_signal_connect (G_OBJECT (button), "clicked",
                  G_CALLBACK (hello), NULL);

/* This will cause the window to be destroyed by calling
 * gtk_widget_destroy(window) when "clicked". Again, the destroy
 * signal could come from here, or the window manager. */
g_signal_connect_swapped (G_OBJECT (button), "clicked",
                          G_CALLBACK (gtk_widget_destroy),
                          G_OBJECT (window));

/* This packs the button into the window (a gtk container). */
gtk_container_add (GTK_CONTAINER (window), button);

/* The final step is to display this newly created widget. */
gtk_widget_show (button);

/* and the window */
gtk_widget_show (window);

/* All GTK applications must have a gtk_main(). Control ends here
 * and waits for an event to occur (like a key press or
 * mouse event). */
gtk_main ();

return 0;
}

```

اصول Callbacks و Signals (سیگنال‌ها و پاسخ‌گوها)

قبل از اینکه جزئیات helloworld را بررسی نماییم، قصد داریم «سیگنال‌ها» و «پاسخ‌گوها» را شرح دهیم، +
یک جعبه ابزار رویدادگرا می‌باشد و این بدین معنی است که بوسیله gtk_main() منتظر می‌ماند تا هنگامی که
رویدادی رخداد و کنترل به تابع مناسب سپرده شود . این عمل یعنی سپردن کنترل با استفاده از «سیگنال‌ها» صورت
می‌گیرد. توجه داشته باشید که این سیگنال‌ها همان سیگنال‌های سیستم یونیکس نیستند ، و توسط آنها پیاده‌سازی
نشده‌اند، اگرچه مجموعه اصطلاحات به کار رفته تقریبا همسان می‌باشند). وقتی که یک رویداد رخداد می‌دهد ، مانند
вшردن دکمه مأوس ، سیگنال مناسب توسط ویجتی که بر روی آن کلیک می‌شود منتشر می‌گردد. این روشی است که +
GTK بیشترین کار مفید خود را بوسیله آن به انجام می‌رساند. سیگنال‌هایی وجود دارند که همه ویجت‌ها آنها را به ارث
می‌برند، مانند «destroy» و همچنین سیگنال‌هایی وجود دارند که مخصوص ویجت خاصی هستند مانند سیگنال
«toggled» در یک دکمه تاگل .

برای ایجاد یک دکمه جهت انجام یک عمل ، ما یک اداره‌گر سیگنال را جهت بدست آوردن این سیگنال‌ها و فراخوانی

تابع مناسب، کار می‌گذاریم. این عمل با تابعی شبیه به این صورت می‌گیرد:

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

در اینجا آرگومان اول(`object`)، نمونه `GtkWidget` می‌باشد که سیگنالی را صادر خواهد نمود و آرگومان دوم (`name`)، یک رشته محتوی نام سیگنالی است که شما می‌خواهید آن را بدست آورید. آرگومان سوم (`func`)، تابعی است که شما می‌خواهید هنگام دریافت سیگنال فراخوانی شود و چهارمین، `func data`، داده‌هایی هستند که شما قصد دارید آن را به این تابع ارسال نمایید. این متدها یک «شناسه اداره‌گر» را برمی‌گردانند که می‌توان از آن جهت قطع یا مسدود کردن اداره‌گر استفاده نمود. تابعی که در آرگومان دوم تعیین شده است یک «تابع پاسخ‌گو» را فراخوانی می‌کند که عموماً به این شکل خواهد بود:

```
void callback_func( GtkWidget *widget,
                     ... /* other signal arguments */
                     gpointer  callback_data );
```

در اینجا اولین آرگومان به ویجتی که سیگنال را صادر کرده، اشاره می‌کند، و `callback data` همان گونه که در بالا دیده شد به داده‌هایی که به عنوان آخرین آرگومان به تابع `(g_signal_connect()` ارسال شده است اشاره می‌کند.

نکته: قالب بالا فقط یک راهنمای عمومی برای اعلان یک تابع پاسخ‌گو به سیگنال می‌باشد، زیرا بعضی از سیگنال‌های خاصی که ویجت‌ها صادر می‌کنند، پارامترهای فراخوانی متفاوتی را به وجود می‌آورند.

فراخوانی دیگری که در مثال `helloworld` از آن استفاده شده تابع ذیل می‌باشد:

```
gulong g_signal_connect_swapped( gpointer          *object,
                                  const gchar    *name,
                                  GCallback      func,
                                  gpointer        *callback_data );
```

در اینجا `g_signal_connect_swapped()` مشابه `g_signal_connect()` می‌باشد به استثنای اینکه وقتی اداره‌گر فراخوانی شود داده و سیگنال منتشر شده معاوضه خواهند شد. بنابراین وقتی این تابع برای اتصال سیگنال‌ها استفاده می‌کنیم، تابع پاسخ‌گو به حالت ذیل خواهد بود:

```
void callback_func( gpointer  callback_data,
                     ... /* other signal arguments */
                     gpointer   *object);
```

در اینجا `object` معمولاً یک ویجت می‌باشد. ما معمولاً تابع پاسخ‌گویی را برای `g_signal_connect_swapped()` هر قدر هم که مهم باشد بربا نمی‌کنیم. هنگامی که یک سیگنال برای اشیا دیگر منتشر می‌شود از این نوع توابع برای فراخوانی توابع GTK که یک ویجت و یا یک شی را به عنوان یک آرگومان می‌پذیرند استفاده می‌شود. در مثال `helloworld` ما سیگنال `clicked` را به یک دکمه متصل کرده‌ایم، اما تابع `gtk_widget_destroy()` برای ویجت `window` فراخوانی می‌شود.

Glade

همان طور که ملاحظه نمودید ایجاد برنامه‌ها با این روش قدری مشکل و وقت‌گیر می‌باشد به همین منظور کتابخانه `glade` ایجاد شده است. برای طراحی با استفاده از این کتابخانه از برنامه‌ای به نام `Glade Interface Designer` استفاده می‌شود. محیط این برنامه بسیار شبیه محیط‌های Visual IDE و یا مانند اکثر برنامه نویسان از یک GUI یا همان ظاهر برنامه ساخته می‌شود و برای کد نویسی بایستی از یک `Glade Interface Designer` (نسخه ۲) این نرم افزار می‌توانست ویرایشگر متن استفاده کنید. در نسخه قبلی `Glade Interface Designer` (نسخه ۱) این نرم افزار می‌توانست خروجی برنامه را علاوه بر قالب XML به صورت کد C و یا C++ تولید نماید ولی به علت وجود نقص در کدهای

تولید شده و نارضایتی کاربران ، در نسخه سوم Glade این قابلیت کنار گذاشته شد و کل پروژه در یک فایل XML ذخیره می شود. در ادامه، مثال قبلی با استفاده از glade نوشته شده است، برای کامپایل این برنامه دستورات ذیل را در خط فرمان تایپ کنید:

```
$ gcc -Wall glade-helloworld.c -o glade-helloworld `pkg-config --cflags --libs
gtk+-2.0 libglade-2.0`  
-----  
  
#include <gtk/gtk.h>  
#include <glade/glade.h>  
  
/* This is a callback function. The data arguments are ignored  
 * in this example. More on callbacks below. */  
static void hello( GtkWidget *widget,  
                   gpointer   data )  
{  
    g_print ("Hello World\n");  
}  
  
static gboolean delete_event( GtkWidget *widget,  
                             GdkEvent  *event,  
                             gpointer   data )  
{  
    /* If you return FALSE in the "delete_event" signal handler,  
     * GTK will emit the "destroy" signal. Returning TRUE means  
     * you don't want the window to be destroyed.  
     * This is useful for popping up 'are you sure you want to quit?'  
     * type dialogs. */  
  
    g_print ("delete event occurred\n");  
  
    /* Change TRUE to FALSE and the main window will be destroyed with  
     * a "delete_event". */  
  
    return TRUE;  
}  
  
/* Another callback */  
static void destroy( GtkWidget *widget,  
                    gpointer   data )  
{  
    gtk_main_quit ();  
}  
  
int main( int   argc,  
          char *argv[] )  
{  
    /* GtkWidget is the storage type for widgets */  
    GtkWidget *window;  
    GtkWidget *button;  
  
    /* GladeXML is the storage type for glade xml file */  
    GladeXML *glade_xml_file;  
  
    /* This is called in all GTK applications. Arguments are parsed  
     * from the command line and are returned to the application. */  
    gtk_init (&argc, &argv);  
  
    /* load the interface */  
    glade_xml_file = glade_xml_new("hw.glade", NULL, NULL);  
  
    /* Fetch window feature from glade file */  
    window = glade_xml_get_widget(glade_xml_file, "window1");
```

```

/* Fetch button feature from glade file */
button = glade_xml_get_widget(glade_xml_file, "button1");

/* Connect signals*/
glade_xml_signal_connect(glade_xml_file, "delete_event", G_CALLBACK
(delete_event));
glade_xml_signal_connect(glade_xml_file, "destroy", G_CALLBACK (destroy));
glade_xml_signal_connect(glade_xml_file, "hello", G_CALLBACK (hello));

/* This will cause the window to be destroyed by calling
 * gtk_widget_destroy(window) when "clicked". Again, the destroy
 * signal could come from here, or the window manager. */
g_signal_connect_swapped (G_OBJECT (button), "clicked",
                           G_CALLBACK (gtk_widget_destroy),
                           G_OBJECT (window));

gtk_widget_show_all(window);

/* All GTK applications must have a gtk_main(). Control ends here
 * and waits for an event to occur (like a key press or
 * mouse event). */
gtk_main ();

return 0;
}

```

برای استفاده از `glade` باستی کتابخانه‌های مربوط به آن نیز بر روی سیستم نصب شده و یا به گونه‌ای در دسترس باشند و علاوه بر این توسعه دهنده‌گان `GTK+` قصد دارند که کتابخانه‌هایی را که توسط طرفهای ثالث (که اکثرا توسعه دهنده‌گان پروژه گنوم می‌باشند) تهیه شده و در بیرون از پروژه `GTK+` قرار دارند و توسط برنامه نویسان استفاده زیادی از آن‌ها می‌شود به پروژه `GTK+` منتقل نمایند و برای این مهم دو نوع راهکار در نظر گرفته‌اند یکی انتقال بی‌نقص آن API‌ها و دیگری دوباره نویسی آن پروژه‌ها در پروژه مادر (`GTK+`)، کتابخانه `glade` نیز از این امر مستثنی نیست و به همین منظور توسعه دهنده‌گان `GTK+` بخشی را با عنوان `GtkBuilder` به این پروژه اضافه نموده‌اند. همچنین `Glade` نیز از نسخه 3.6 به بعد از این فرمت پشتیبانی می‌نماید. در ادامه مثال قبلی با استفاده از `GtkBuilder` نوشته شده است، برای کامپایل این برنامه دستورات ذیل را در خط فرمان تایپ کنید:

```
$ gcc -Wall gtkbuilder-helloworld.c -o gtkbuilder-helloworld `pkg-config
--cflags --libs gtk+-2.0`
```

```
-----
#include <gtk/gtk.h>

/* This is a callback function. The data arguments are ignored
 * in this example. More on callbacks below. */
static void hello( GtkWidget *widget,
                   gpointer   data )
{
    g_print ("Hello World\n");
}

static gboolean delete_event( GtkWidget *widget,
                             GdkEvent  *event,
                             gpointer   data )
{
    /* If you return FALSE in the "delete_event" signal handler,
     * GTK will emit the "destroy" signal. Returning TRUE means
     * you don't want the window to be destroyed.
     * This is useful for popping up 'are you sure you want to quit?'
     * type dialogs. */
}
```

```

g_print ("delete event occurred\n");

/* Change TRUE to FALSE and the main window will be destroyed with
 * a "delete_event". */

return TRUE;
}

/* Another callback */
static void destroy( GtkWidget *widget,
                     gpointer   data )
{
    gtk_main_quit ();
}

int main( int   argc,
          char *argv[] )
{
    /* GtkWidget is the storage type for widgets */
    GtkWidget *window;
    GtkWidget *button;

    /* GtkBuilder is the storage type for ui xml file */
    GtkBuilder *uixml;

    /* This is called in all GTK applications. Arguments are parsed
     * from the command line and are returned to the application. */
    gtk_init (&argc, &argv);

    /* load the interface */
    uixml = gtk_builder_new();
    gtk_builder_add_from_file(uixml, "hw.ui", NULL);

    /* Fetch window feature from glade file */
    window = GTK_WIDGET(gtk_builder_get_object (uixml, "window1"));
    /* Connect signals*/
    g_signal_connect(G_OBJECT(window), "delete-event", G_CALLBACK(delete_event),
NULL);
    g_signal_connect(G_OBJECT(window), "destroy", G_CALLBACK(destroy), NULL);

    /* Fetch button feature from glade file */
    button = GTK_WIDGET(gtk_builder_get_object (uixml, "button1"));
    /* Connect signals*/
    g_signal_connect(G_OBJECT(button), "clicked", G_CALLBACK(hello), NULL);

    /* This will cause the window to be destroyed by calling
     * gtk_widget_destroy(window) when "clicked". Again, the destroy
     * signal could come from here, or the window manager. */
    g_signal_connect_swapped (G_OBJECT (button), "clicked",
                             G_CALLBACK (gtk_widget_destroy),
                             G_OBJECT (window));

    gtk_widget_show_all(window);

    /* All GTK applications must have a gtk_main(). Control ends here
     * and waits for an event to occur (like a key press or
     * mouse event). */
    gtk_main ();

    return 0;
}

```

منابع :

<http://www.wikipedia.org>

<http://www.gtk.org>