Journal of Computer Applications
计算机应用,2013,33(4):1146 - 1148
ISSN 1001-9081
CODEN JYIIDU
2013-04-01
http://www.joca.cn

文章编号:1001 - 9081(2013)04 - 1146 - 03
doi:10.3724/SP.J.1087.2013.01146

基于客户端软件渲染优化 Wine 图形性能的方法

黄聪会[1*],陈　靖[1],朱清超[1],郭威武[2]
(1.空军工程大学 信息与导航学院,西安 710077;　2.93010 部队,沈阳 110015)
(*通信作者电子邮箱 huangdoubleten@126.com)

摘　要:针对 Wine 操作设备无关位图(DIB)存在性能瓶颈的问题,提出一种客户端软件渲染的方法。该方法首先分析操作 DIB 的 GDI API 函数,然后确定客户端软件渲染的加载点,再以链表形式将不同设备上下文环境及其对应的 GDI API 函数串联,在此基础上实现 GDI API 函数的客户端软件渲染。性能测试表明,经客户端软件渲染优化后的 Wine 与未优化的 Wine 相比,其操作 DIB 的性能平均至少提高了 10 倍,且接近本地 Windows XP 下操作 DIB 的性能,有效地避免了操作 DIB 的性能瓶颈。

关键词:虚拟化技术;软件渲染;图形性能优化;设备无关位图;WINE
中图分类号:TP311　　**文献标志码**:A

Graphics performance optimization method for Wine based on client software rendering

HUANG Conghui[1*], CHEN Jing[1], ZHU Qingchao[1], GUO Weiwu[2]
(1. *Information and Navigation Institute, Air Force Engineering University, Xi'an Shaanxi* 710077, *China*;
2. 96201 *Army, Shenyang Liaoning* 110015, *China*)

Abstract: To deal with the performance bottleneck of operating the Device Independent Bitmap (DIB) in Wine, a method of client software rendering was brought forward. This method firstly analyzed GDI API for operating DIB, and then confirmed the load point of client software rendering, subsequently designed a list for linking different device context and their corresponding GDI API, at last realized the client software rendering of GDI API. The performance test shows that compared with the Wine without optimization, the average graphic performance of the Wine with the optimization of client software rendering enhances at least 10 times when it operates DIB, and is close to the performance of the local Windows XP, which effectively avoids the performance bottleneck of operating DIB.

Key words: virtualization technology; software rendering; graphic performance optimization; Device Independent Bitmap (DIB); Wine

0　引言

X 窗口系统是一种基于网络的 GUI 系统,是当前大多数 UNIX 和 Linux 操作系统上的图形用户界面[1]。在实现上 X 窗口系统采用了 C/S 架构模型,由一个 X 服务器与多个 X 客户端程序进行通信,服务器反馈用户输入(键盘、鼠标、触摸屏),并接受客户端对图形输出(窗口)的请求。而 Wine 作为一种支持 Windows 程序运行于 Linux 上的进程虚拟机[2],其图形处理能力必然受制于 X 窗口系统,因此相对 Windows 操作系统而言总体性能不高[3];而且 Windows 图形系统与 X 窗口系统的异构将导致 Wine 在模拟 Windows 图形 API 时存在性能瓶颈。当 Wine 遭遇此类性能瓶颈时,其图形性能将急剧恶化,严重影响用户的使用体验。

Wine 存在多个相关的图形性能瓶颈,如应用程序频繁地编译 Direct3D 着色器将导致很低帧率(Frame Per Second, FPS),应用程序严重依赖内核事件将运行缓慢等。此类瓶颈是由 Wine 模拟 Windows 功能时的不合理设计导致的。通过改善或优化 Wine 中对应功能模块的设计,可消除此类图形性能瓶颈[4]。

本文在分析 Windows 操作系统的窗口系统与 X 窗口系统差异的基础上,给出了 Wine 在 X 窗口系统基础上模拟 Windows 图形 API 所产生的一个性能瓶颈,并提出了相应的解决方案。

1　Wine 图形性能瓶颈分析

在桌面操作系统中,当前存在两种主要的窗口系统。一种是基于 C/S 架构,将窗口系统核心作为操作系统的服务器用户进程,同时将窗口系统的应用程序作为客户端用户进程,通过进程间通信方式,由窗口服务器进程实现窗口核心功能;X 窗口系统即为此类窗口系统的典型[5]。另一种则是基于核心的窗口系统,即把窗口系统的核心放入操作系统的内核中。用户以系统调用的方式使用窗口功能;微软 Windows 即为此类窗口系统的典型。基于 C/S 架构的窗口系统具有较高的稳定性,即该窗口系统的崩溃不会引起底层操作系统的崩溃,通过重启服务器进程可自动恢复,但由于使用过程涉及多次进线程切换,因此此类窗口系统效率较低[6]。而基于核心的窗口系统不需要进线程切换,因此效率较高,但该窗口系统的崩溃将导致整个操作系统的崩溃,因此稳定性较差。

Wine 通过模拟 Windows API 并提供虚拟运行环境,以支持 Windows 程序在 Linux 上的运行。当模拟 Windows GDI API 时,Wine 基于 X 窗口系统提供的 Xlib 库进行仿真。由于基于 C/S 架构的 X 窗口系统与基于核心的微软 Windows 窗口系统存在较大差异,由此将造成 Wine 在操作 DIB(Device

收稿日期:2012-10-23;**修回日期**:2012-12-02。　**基金项目**:国家自然科学基金资助项目(61172083)。
作者简介:黄聪会(1985 -),男,湖南衡阳人,博士研究生,主要研究方向:虚拟化;　陈靖(1963 -),女,山西洪桐人,教授,博士生导师,主要研究方向:分布式系统;　朱清超(1988 -),男,山东济宁人,博士研究生,主要研究方向:分布式系统;　郭威武(1983 -),男,辽宁沈阳人,助理工程师,硕士,主要研究方向:计算机网络。

Independent Bitmap)位图时出现性能瓶颈。具体分析如下:

在 Windows 操作系统的 GDI 子系统中,存在两种位图对象,分别为 DDB(Device Dependent Bitmap)位图和 DIB[7]。其中:DDB 位图的数据格式与设备相关,Windows 应用程序只能通过 GDI API 访问;DIB 的数据格式独立于设备,既可通过 GDI API 访问,也可直接内存访问。当在 Wine 中模拟 Windows 程序操作 DIB 时,Wine 即相当于发起 X 请求的客户端,其处理过程如图 1 所示。

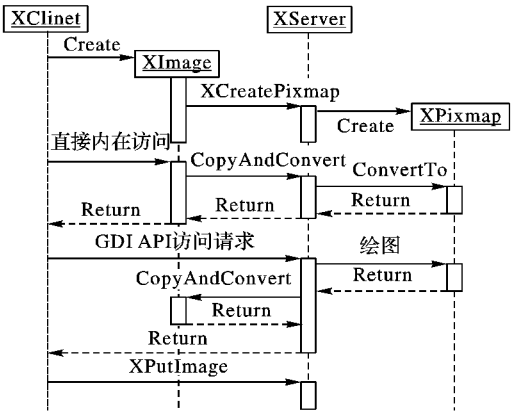

图 1　Wine 中操作 DIB 的序列图

为支持 Windows 程序对 DIB 的操作,Wine 在 X 客户端创建 XImage 结构(相当于 DIB)和 X 服务器创建 XPixmap 结构(相当于 DDB 位图),同时保持两种结构之间的自动同步,以模拟 DIB 并支持对它的两种不同操作。当模拟 Windows 程序直接内存访问 DIB 时,相当于修改 XImage。修改后的 XImage 将自动复制到 X 服务器,并转换为对应的 XPixmap。当模拟 Windows 程序通过 GDI API 操作 DIB 时,相当于修改 XPixmap。修改后的 XPixmap 将自动复制到 X 客户端,并转换为对应的 XImage。

该处理过程对 DrawArc、LineTo、SetPixel、GetPixel 等 GDI API 均成立。这种方法模拟 GDI API 操作 DIB 简单有效,但仍存在一些问题:

1)XImage 与 XPixmap 之间的同步将花费巨大代价。随着同步次数的增多,将形成性能瓶颈。

2)XPixmap 必须与 X 服务器所支持的位图格式一致。因此当 X 服务器只支持 16 bit/pixel 位图时,X 客户端传输给 X 服务器端的 32 bit/pixel 位图将不得不转换为 16 bit/pixel 位图,而在传送回 X 客户端时,则必须将 16 bit/pixel 位图转换为 32 bit/pixel 位图[8]。显然该处理过程中将存在颜色分辨率的损失。

2　增强 Wine 图形性能的方法

从上述对 Wine 图形性能瓶颈的分析可知,当应用程序在使用 GDI API 和直接内存访问修改 DIB 间频繁地切换时,将造成 Wine 图形性能的急剧下降。而解决该瓶颈的方法是采用软件渲染技术在 X 客户端仿真 GDI API 操作 DIB,从而避免 DIB 在 X 服务器端和客户端间来回传输的巨大代价。

2.1　基本原理

在计算机图形学中,渲染通常指运用解析几何的方法计算场景中的物体到一个称为投影平面的面上形成的投影,这种方式与相机将场景投影到胶卷上的方式类似。渲染可分为硬件渲染和软件渲染[9]。硬件渲染即通过硬件加速实现渲染流程,具有速度快的优点,但需要图形显示卡的支持;软件渲染即通过软件的方式实现渲染流程,速度较慢,但不需要图形显示卡的支持[10]。当前 Wine 中模拟 GDI API 操作 DIB 即是一种软件渲染。它通过将 GDI API 转换为 X 请求,然后 X 服务器根据该请求修改 DIB,最后将修改后的 DIB 传送回客户端[11]。此类软件渲染存在于 X 服务器,而从前述分析可知,这种方法将造成 Wine 图形性能瓶颈,而解决方法是在 X 客户端实现 GDI API 的软件渲染,如图 2 所示。

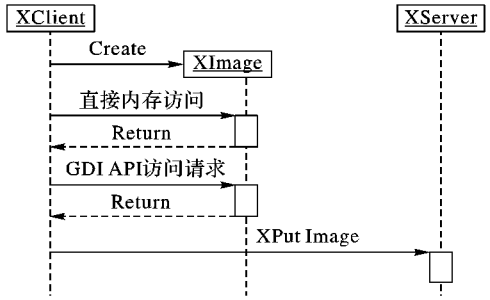

图 2　客户端软件渲染方法

Wine 作为 X 客户端,将创建 XImage 结构以仿真 DIB。当 Windows 程序调用 GDI API 操作 DIB 时,Wine 将在 X 客户端软件渲染 GDI API,并将修改结果写入 XImage 结构。这将完全避免在 GDI API 和直接内存访问修改 DIB 间频繁切换引起的巨大负载。

2.2　方法分析

为保持开源软件 Wine 源代码的健壮性和风格的一致性,客户端软件渲染方法在 Wine 中的设计实现应满足以下两个条件:

1)该方法应以一种增量的、非侵入的方式集成到当前 Wine 的代码库中。

2)当客户端软件渲染方法因各种不可预料的原因而失败时,GDI API 操作应能继续以原有方式模拟实现。

根据此设计要求,在 X 客户端实现 GDI API 的软件渲染需解决以下三个问题:

1)需在 X 客户端渲染哪些 GDI API;

2)在何处加载 GDI API 的软件渲染代码;

3)当加载 GDI API 软件渲染代码失败或软件渲染失败时,如何继续使用原有方法。

针对第一个问题,通过分析可知,虽然存在几百个 GDI API,但实际上在不同的设备上下文(Device Context,DC)环境下,能够使用的 GDI API 函数十分有限且各不相同[12]。操作 DIB 通常在内存 DC 下,而在内存 DC 下操作 DIB 的相关 GDI API 函数只有 33 个。这些 GDI API 函数可分为两大类:其中一类与在 DIB 上绘制图形或字体有关,如 LineTo、Pie、Ellipse、ExtTextOut 等;另一类则与操作 GDI 对象相关,如 SelectPen、CreateDC 等。

针对后两个问题,应分析在 Windows 程序中如何操作 DIB。在使用 DIB 之前应创建 DIB 对象。而获得 DIB 句柄的所有函数,都最终调用 CreateDIBSection 函数创建 DIB 对象。通过调用 CreateDIBSection 函数,不仅为 DIB 对象分配了内存空间,并返回 DIB 句柄以及直接操作 DIB 内存空间的指针。通过将该 DIB 句柄用 SelectObject 函数选入一个内存 DC,就可使用 GDI API 函数修改 DIB。

2.3　关键数据结构及其应用

通过 Windows 程序操作 DIB 的行为分析可知,是否对 GDI API 函数进行客户端软件渲染由 GDI API 函数使用的设备上下文决定。当 GDI API 函数接受一个内存 DC 作为其参数时,就应启动客户端软件渲染。为此设计如下的数据结构 gdi_physdev,该数据结构拥有三个成员,其中 hdc 是设备上下文的句柄,而 funcs 则是在 hdc 所指定的设备上下文下能够使

用的函数，next 则是指向下一个 gdi_physdev 的指针。

```
typedef struct gdi_physdev
{
    const struct gdi_dc_funcs  * funcs;
    struct gdi_physdev         * next;
    HDC                          hdc;
} * PHYSDEV;
```

通过数据结构 gdi_physdev，可将不同的设备上下文及其对应的操作函数用链表链接起来，从而解决加载 GDI API 的软件渲染代码，以及加载失败或渲染失败时继续使用原有方法的设计问题。通常在 GDI API 函数 CreateCompatibleDC 中创建与指定设备兼容的内存 DC，并返回内存 DC 对象的句柄。当模拟该函数时将生成如图 3 所示 gdi_physdev 链表。

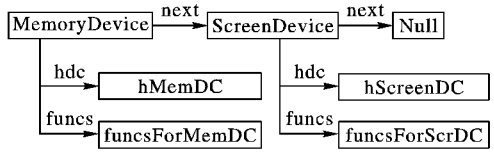

图 3 gdi_physdev 链表

创建内存 DC 对象后，即可将 DIB 句柄用 SelectObject 函数选入，然后调用 GDI API 对 DIB 进行操作。以 GDI API 函数 LineTo 为例，说明如何模拟实现 LineTo 函数，并操作 DIB。模拟 LineTo 函数实现的控制流图如图 4 所示。

当操作 DIB 时，LineTo 函数的输入是内存 DC 的句柄，以及线段终点的横坐标 x 和纵坐标 y。根据内存 DC 的句柄，首先将获取上述 CreateCompatibleDC 函数创建的 gdi_physdev 链表的表头指针；随后即可根据该表头指针调用内存 DC 对应的 LineTo 函数，即采用了客户端软件渲染的 LineTo 函数。如果函数调用成功，即返回 true；否则链表指针将指向下一个元素，再进行上述尝试，即采用原有方法进行实验，直到链表尾，然后退出。

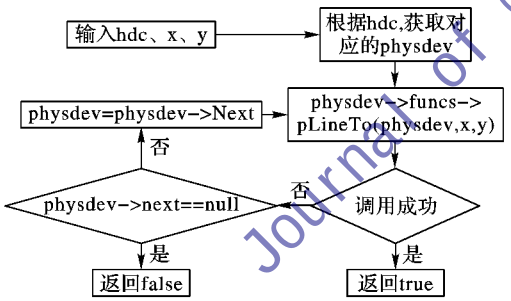

图 4 LineTo 控制流图

3 性能测试及结果分析

为验证客户端软件渲染方法对 Wine 图形性能的改进，将采用微基准程序进行测试[13]。测试计算机硬件配置为 Inter Core2 T6400 2G Hz、DDR Ⅱ 2 GB 内存、NVIDIA GeForce 9300 MHz显卡、250 GB 硬盘、千兆网卡。软件配置为 Ubuntu 10.04、Wine 1.3.17。

3.1 测试方法及结果

微基准程序采用 Visual C ++ 6.0 开发。具体操作是选取一种 GDI API 操作 DIB，然后采用直接内存访问的方式设置 DIB 中一个像素的颜色。选取的 GDI API 如图 5 横坐标所示。微基准程序将给出前述操作所需时间。采用该微基准程序测试 Wine、客户端软件渲染优化后的 Wine 以及 Windows XP SP3 上的运行时间，并以未优化的 Wine 运行时间为基准，给出优化后的 Wine 以及 Windows XP SP3 的时间加速比，如图 5 所示。

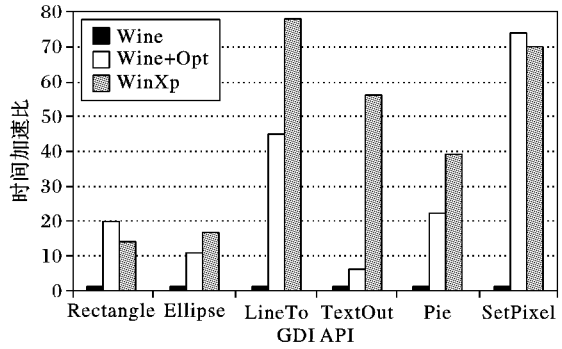

图 5 修改 DIB 的时间加速比

3.2 结果分析

从图 5 中可知，经客户端软件渲染优化后的 Wine 与未优化的 Wine(Wine + Opt)相比，其操作 DIB 的性能平均至少提高了 10 倍。而且这仅是一次 GDI 和内存操作切换的时间加速比，当 Windows 应用程序频繁地在 GDI 和内存模式间切换时，优化后的 Wine 的图形性能优势将更加明显。此外，从图 5 中可分析出优化后的 Wine 操作 DIB 的图形性能已接近在本地 Windows XP 下操作 DIB 的性能。

4 结语

Wine 操作 DIB 形成的性能瓶颈严重影响用户的体验。针对该问题，本文提出一种在 X 客户端实现 GDI API 软件渲染的方法，避免了应用程序操作 DIB 时在 GDI 模式和内存模式之间频繁切换所带来的巨大负载。通过巧妙的设计，该方法可以一种增量的、非侵入的方式集成到当前 Wine 的代码库中，并当客户端软件渲染方法因各种不可预料的原因而失败时，GDI API 操作能继续用原有方式模拟实现。最后，性能测试表明，该方法能显著提升 Wine 操作 DIB 的图形性能，有效地避免操作 DIB 的性能瓶颈。

参考文献：

[1] 樊葆华，窦文华. X 窗口系统中的实时视频图像显示技术研究[J]. 计算机工程与科学，2005，27(2)：39－41.

[2] 王亚军，刘金刚. Windows 程序运行于 Linux 系统的技术[J]. 计算机应用，2009，29(8)：2128－2131.

[3] 黄聪会，陈靖，张黎，等. 兼容 Windows 程序的 KgdLinux 操作系统研究[J]. 空军工程大学学报：自然科学版，2012，13(5)：55－59.

[4] Wine Wiki. Performance[EB/OL]. [2012－10－24]. http://wiki.winehq.org/Performance.

[5] 高二辉，朱建良. 基于 X 窗口系统的中文 TrueType 字体应用研究[J]. 计算机工程，2008，34(4)：104－106.

[6] FENG YUAN. Windows 图形编程[M]. 英宇工作室，译. 北京：机械工业出版社，2002.

[7] 陈文涛. 图像检测系统的算法接口设计与实现[J]. 重庆工商大学学报：自然科学版，2007，24(4)：336－341.

[8] 岳玉芳，陶应学，李有宽，等. 自定义位图在激光传输仿真中的应用[J]. 计算机工程与应用，2005，41(3)：202－204.

[9] MILEFF P，DUDRA J. Modern software rendering[J]. Production Systems and Information Engineering，2012，6(3)：55－66.

[10] MILEFF P，DUDRA J. Efficient 2D software rendering[J]. Production Systems and Information Engineering，2012，6(5)：99－110.

[11] LEWYCKY A，STATE G. DirectX and Wine[EB/OL]. (2001－05－59)[2012－10－24]. http://lwn.net/2001/features/OLS/pdf/pdf/directx.pdf.

[12] 肖习攀，阎小兵，贾迎乐，等. GUI 应用程序移植[M]. 北京：电子工业出版社，2007.

[13] 吴俊峰，戴桂兰，白晓颖，等. 桌面操作系统性能测试研究[J]. 计算机科学，2006，33(9)：257－261.