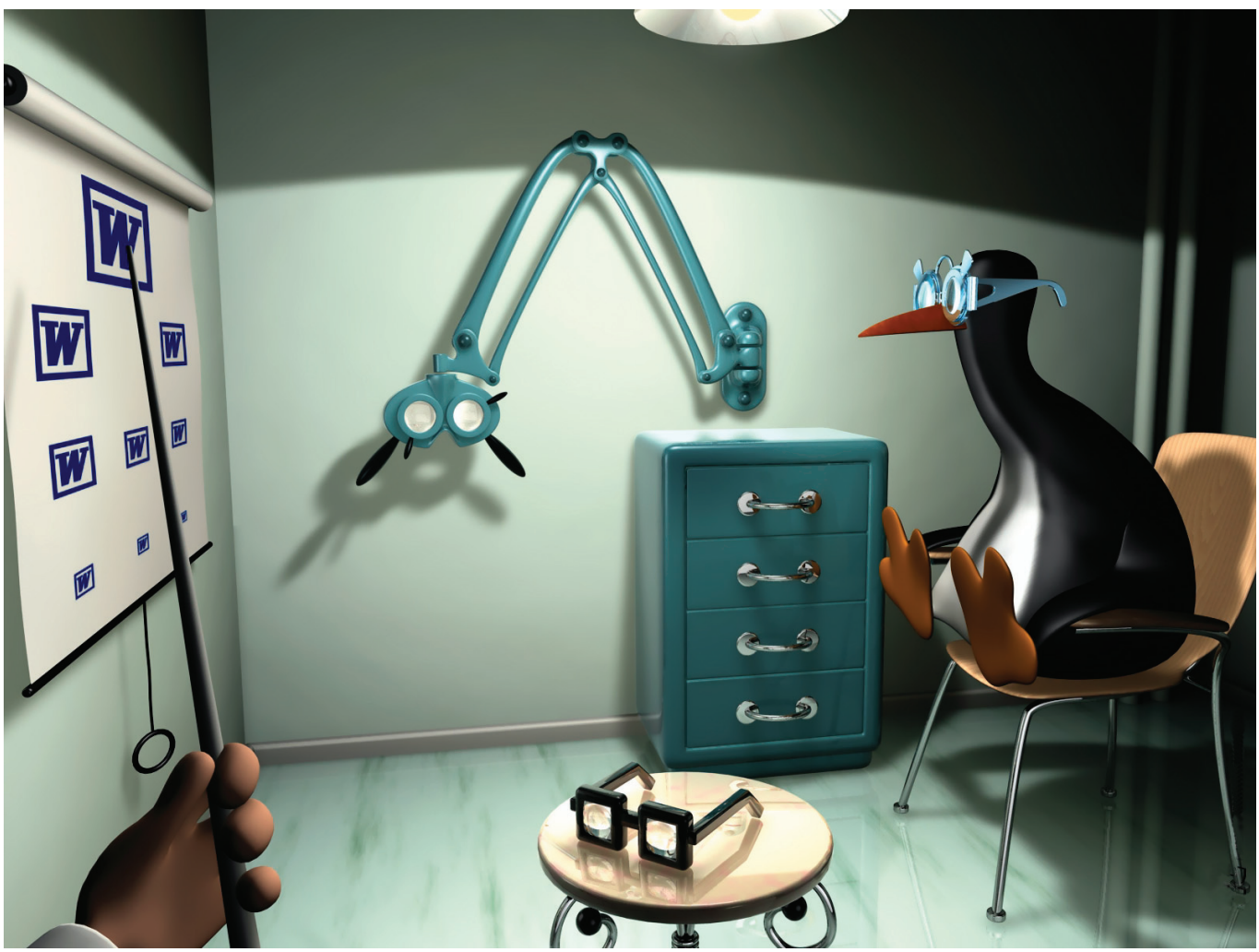
ЗАПУСК WINDOWS-ПРИЛОЖЕНИЙ ПОД LINUX С ПОМОЩЬЮ CROSSOVER OFFICE ЧАСТЬ 2



В первой статье о CrossOver Office мы обсудили, как выполнить инсталляцию этого программного пакета. Затем поговорили о том, как с помощью него запускать под управлением Linux программы, написанные для Windows. В качестве примера было рассказано об успешной работе с Microsoft Office 2000, The Bat!, Microsoft Internet Explorer, Mplayer и Outlook Express. Тем, кто пропустил первую статью, рекомендую обязательно ознакомиться с ней. Сделать это можно либо в февральском выпуске этого журнала, либо на моем сайте http://onix.openet.ru. Ну а я потихоньку продолжу линию нашего повествования.

АНДРЕЙ БЕШКОВ

Задача на сегодня довольно проста. Нужно изучить теорию функционирования и глубинные механизмы CrossOver Office. В дальнейшем это поможет нам правильно идентифицировать проблемы, возникающие при инсталляции Windows-программ, а, как говорят врачи, правильный диагноз – половина лечения. Как было отмечено в первой статье, все Windows-программы делятся на два вида. Официально поддерживаемое обеспечение скорее всего легко установится и будет гладко работать под управлением CrossOver Office, навевая на вас мысли о невыносимой легкости бытия Linux-пользователя. К сожалению, список этих программ не так уж велик. Ознакомиться с ним можно по следующему адресу: http://www.codeweavers.com/site/compatibility/browse/cat.

Кстати, стоит отметить, что в него входят только те приложения, за стабильную работу которых сотрудники компании Codeveawers могут поручиться на все сто процентов. К сожалению, сотрудников в компании не так уж и много, поэтому они не могут знать о всех используемых вами Windows-приложениях. Впрочем, даже если они и будут знать, это не очень изменит положение вещей. Ведь на тестирование каждого приложения нужно потратить немало времени. Таким образом, получается, что приложения, не подвергшиеся тестированию под CrossOver Office, не получают официальной поддержки. Ничего страшного в этом нет. Возможно, при работе с нашими Windows-программами все будет хорошо, и нам никогда не придется прибегать к поддержке персонала Codeveawers. Например, я довольно легко установил Remote Administrator, QuickTime Player, Acrobat Reader и Microsoft Visio и весьма успешно работаю с этими приложениями. Все вышесказанное означает только то, что, начав работать с неподдерживаемыми программами, надеяться нам не на кого и мы, как кошки, начинаем гулять сами по себе. Впрочем, это ни в коем случае не должно страшить нас.

В связи с тем, что CrossOver Office основан на коде, унаследованном от Wine, практически все, о чем я буду рассказывать в этой статье, может вполне успешно применяться при работе с Wine. Итак, начнем с самых основ и посмотрим, что происходит в системе во время запуска Windows-приложений. Сам по себе процесс загрузки программы в память несложен. Проблема в том, что нужно найти и загрузить все требуемые DLL. Затем провести импортирование из DLL требуемых функций и правильно найти точки входа в каждую из них. Предполагается, что каждое нормально написанное приложение, как и любые небазовые DLL, не станет пытаться использовать напрямую системные функции, а вместо этого будет импортировать все требуемые механизмы из основных системных библиотек. Соответственно для того, чтобы Wine смогла нормально загружать Windows-приложения, нужно как минимум заменить своими собственными реализациями основные системные DLL, в число которых входят USER/USER32, GDI/GDI32, KERNEL/KERNEL32 и NTDLL.

Кстати, стоит отметить, что многие функции, реализованные изначально в KERNEL32 и ADVAPI32, постепенно были перенесены в NTDLL. Динамические библиотеки, заново реализованные для Wine, называются «встроенными», а те, что перенесены из Windows без каких-либо изменений, называются «родными». Соответственно в тот момент, когда Windows-приложение заявляет о своем желании импортировать какую-либо DLL, Wine сначала смотрит в список встроенных DLL, если там найти нужное не удалось, то начинается поиск на жестком диске родного Windows DLL-файла.

Сердцем системы эмуляции служит Wine-сервер, если говорить по существу, то он необходим для правильной организации межпроцессорных коммуникаций между всеми запущенными Windows-приложениями и является отдельным однопоточным процессом с бесконечным циклом выборки и обработки сообщений, приходящих от системы и клиентских процессов. Из-за отсутствия многопоточности внутри своего процесса Wine-сервер не может выполнять задачи, требующие существенных временных затрат. Впрочем передача сообщений и событий между клиентскими процессами и основной системой действительно не является трудоемкой работой. Также в целях безопасности Wine-сервер не имеет доступа к адресному пространству своих клиентов. Если в момент старта первого клиентского Wine-процесса процесс Wine-сервера еще не работает, то он будет незамедлительно запущен. Сразу же после начала работы Wine-сервер создает UNIX-сокет, предназначенный для общения с клиентами, отвечающими за выполнение Windows-приложений. Обычно сокет находится в директории $HOME/.wine либо там, куда указывает переменная окружения WINEPREFIX. Ну а если мы используем CrossOver Office, то сокет возникнет в директории /tmp/wine-<имя пользователя>. После того как все клиентские Wine-процессы будут завершены, закончит свою работу и Wine-сервер. Методы работы с приложениями win32 и win16 довольно сильно отличаются. Для программ с архитектурой win16, доставшихся нам в наследство от старых версий Windows, характерно функционирование в едином адресном пространстве и использование кооперативной многозадачности. Исходя из таких требований, удобнее всего было сделать так, чтобы все запущенные win16-программы выполнялись не как отдельные процессы, а стали нитями в рамках одного процесса. Идея подобного механизма работы с устаревшими приложениями была позаимствована из Windows NT, поэтому так же, как и в системе-прародителе, сущность, обеспечивающая функционирование приложений win16, называется WOW-процессом. Внутри него разные нити, олицетворяющие отдельные приложения, синхронизируются друг с другом с помощью мьютекса. Нить, работающая в данный момент, захватывает мьютекс в эксклюзивное владение и соответственно не позволяет работать остальным нитям. Как только активная нить посчитает, что выполнила достаточно действий, мьютекс будет освобожден и начнет подавать сигналы, указывающие на его холостятский статус. В результате этого он будет вновь захвачен следующей нитью, стоящей в очереди на выполнение. Таким образом реализуется кооперативная многозадачность. Ну а WOW-процесс будет существовать до тех пор, пока внутри него работает хотя бы одна нить. Приемы, используемые для работы с win32-программами, выглядят совершенно иначе. Для каждого выполняемого внутри Wine приложения win32 создается свой собственный процесс, так как программы такого типа написаны в расчете на вытесняющую многозадачность и отдельное виртуальное адресное пространство для каждого экземпляра. Каждый клиентский процесс Wine имеет в своем составе специальную сервисную нить, выполняющуюся наравне с остальны-

ми нитями этого процесса. Ее функция состоит в том, чтобы выполнять задачи, которые нельзя поручить Wine-серверу. Например, с помощью него удается реализовать для каждого процесса отдельную очередь ожидания событий, исходящих от системы и других Wine-процессов. По приходу события сервисная нить процесса выходит из состояния сна, обрабатывает поступившее событие и передает сообщение о нем другим нитям процесса. Любые компоненты ядра wine могут устанавливать свои собственные обработчики в сервисную нить, если есть необходимость выполнять какие-либо действия вне зависимости от цикла выборки сообщений эмулируемого приложения.

Следующей проблемой является поддержка реестра. Ведь в любой, даже свежеустановленной Windows-системе в реестре уже записано довольно большое количество данных, поэтому нам необходимо иметь свою собственную реализацию реестра. В отличие от Windows, где реестр хранится в бинарном формате, wine использует для этой цели обычный текстовый файл. Соответственно ключи реестра записываются примерно в таком формате:

```
[AINF0008\\0.map] 1069188705
"008cc782b199a527"=",33,HKCR,Interface\\ ↵
    {56a868b5-0ad4-11ce-b03a-0020af0ba770}\\Distributor,,"i
"140e1cba790e4932"=",33,HKLM,Software\\Microsoft\\ ↵
    Multimedia\\DirectXMedia,.Prog,"
```

А это значит, что в случае необходимости мы легко сможем добавлять или удалять данные в этот импровизированный реестр с помощью обычного текстового редактора. Текстовый формат хранения данных не единственное отличие реестра, используемого Wine, от настоящего Windows-реестра. Дело в том, что под Wine реестр хранится не в одном, а сразу в нескольких отдельных файлах. Сразу же после инсталляции CrossOver Office ветви реестра, обеспечивающие базовую функциональность Windows, будут записаны в файл system.reg, находящийся в директории $HOME/.cxoffice/dotwine. Дабы не испортить системный реестр, все изменения, вносимые в него во время установки и последующего жизненного цикла пользовательских программ, записываются в отдельный файл user.reg.

Следующим условием, необходимым для правильного функционирования Windows-программ, является наличие на диске строго определенной иерархии директорий. С точки зрения Windows наши директории должны выглядеть так.

```
C:\
├─ Program Files
│   ├─ Accessories
│   ├─ Common Files
│   └─ Netscape
├─ Recycled
├─ Temp
└─ Windows
    ├─ All Users
    ├─ Application Data
    ├─ Command
    ├─ Cookies
    ├─ Desktop
    ├─ Favorites
    ├─ Fonts
    ├─ Help
    ├─ Icons
    ├─ NetHood
    ├─ Recent
    ├─ SendTo
    ├─ ShellNew
    ├─ Start Menu
    ├─ System
    ├─ System32
    └─ Temp
```

CrossOver Office хранит все эти директории в $HOME/.cxoffice/dotwine/fake_windows/. Создаются они автоматически и так же легко наполняются всеми необходимыми файлами сразу же после запуска программы /opt/cxoffice/bin/officesetup.

Вот теперь, когда мы уже разобрались со многими нюансами работы эмулятора, пришло время посмотреть, как выглядит изнутри главный конфигурационный файл. Для Wine этот файл называется $HOME/.wine/wine.conf, а для CrossOver соответственно $HOME/.cxoffice/dotwine/config. Формат обоих файлов почти ничем не отличается в обеих реализациях. Первые несколько секций отвечают за определение виртуальных дисков для Windows-системы. Проблема состоит в том, что в UNIX-подобных операционных системах, в отличие от Windows, файловая система не разбита на отдельные диски, обозначаемые латинскими буквами от A до Z, а представлена единым деревом. Ну а вторая досадная неувязка кроется в том, что традиционная файловая система Windows не различает строчные и прописные буквы в именах файлов и директорий. Соответственно файл с именем MyDOCUMENT и mydocument для Windows являются одним и тем же. Таким образом, перед нами стоит задача сделать так, чтобы Windows считала некоторые директории файловой системы UNIX своими виртуальными дисками. Кстати, стоит отметить, что те строки, которые начинаются знаком «;», воспринимаются эмулятором как комментарии.

```
# Имя виртуального диска. В данном случае это диск A:
[Drive A]
# Путь к директории, которая хранит в себе файлы. По умолчанию
# скрипт инсталляции вписывает сюда директорию /mnt/floppy/,
# но для меня это не подходит в связи с тем, что в моей
# системе работает демон autofs, который монтирует устройство
# дисковода /dev/floppy в директорию /mnt/floppy/auto/.
# Поэтому пришлось поправить путь вручную.
"Path" = "/mnt/floppy/auto"
# Тип устройства. Думаю, всем понятно, что для Windows оно
# будет выглядеть как гибкий диск. Эта переменная может
# принимать значения hd, cdrom, network, floppy соответственно
# обозначающие жесткий диск, CD-ROM, сетевая папка,
# гибкий диск.
"Type" = "floppy"
# Метка тома. Обычно нужна только для программ, которые
# пытаются защищаться от копирования с помощью чтения этой
# метки.
"Label" = "Floppy A"
# Имя физического устройства обычно используется для
# низкоуровневого чтения и записи. Эту опцию можно применять
# только к гибким дискам и CD-ROM.
# Если попытаться применить такую опцию к любому другому
# устройству, то результаты будут очень неприятные.
# Низкоуровневая запись скорее всего повредит родную UNIX
# файловую систему.
"Device" = "auto"

# Описание диска C:
 [Drive C]
"Path" = "fake_windows"
# Судя по типу устройства, Windows будет считать, что это
# жесткий диск.
"Type" = "hd"
"Label" = "fake_windows"
# Тип файловой системы,  поведению которой Wine будет
# подражать. Может принимать значения: win95, msdos, unix.
# Соответственно в случае опции win95 эмулируемая система будет
# думать, что это FAT32 с поддержкой длинных имен. Регистр
# символов в именах файлов не различается. Если использовать
# тип msdos, то на имена файлов накладывается ограничение,
# соответствующее файловой системе MS-DOS. Опция unix
# отображает файловую систему точно так же, как она выглядит
# для UNIX. Эта опция практически не применятся, так как
# Windows не умеет работать с такими файловыми системами.
"Filesystem" = "win95"
```

```
# Опция, отвечающая за перекодирование имен файлов
# и директорий в родную кодировку.
"Codepage" = "0"

# Описание диска M:
 [Drive M]
# Это первый CD-ROM в системе. Как обычно, из-за демона
# autofs пришлось поменять путь с /mnt/cdrom на /mnt/cdrom/auto
"Path" = "/mnt/cdrom/auto"
"Type" = "cdrom"
"Label" = "CD-ROM M"
"Filesystem" = "win95"
"Device" = "auto"

# Описание диска N:
 [Drive N]
# Второй CD-ROM. Все установки сделаны аналогично диску M.
"Path" = "/mnt/cdrom1/auto"
"Type" = "cdrom"
"Label" = "CD-ROM N"
"Filesystem" = "win95"
"Device" = "auto"

# Описание диска Y:
[Drive Y]
# Здесь в качестве диска Y монтируется домашняя директория
# пользователя.
"Path" = "%HOME%"
# А вот такой тип устройств в стандартном Wine не встречается,
# он характерен только для CrossOver Office. Впрочем, для Windows
# это устройство все равно выглядит, как обычный жесткий диск.
"Type" = "%CXOFFICE_DRIVE_TYPE_HACK%"
"Label" = "Home"
"Filesystem" = "win95"
"Codepage" = "0"

# Описание диска Z:
 [Drive Z]
# А тут монтируется корень всей файловой системы.
"Path" = "/"
"Type" = "%CXOFFICE_DRIVE_TYPE_HACK%"
"Label" = "Root"
"Filesystem" = "win95"
"Codepage" = "0"
```

Разобравшись с виртуальными дисками, идем дальше. Следующая интересная для нас секция выглядит так:

```
[wine]
# Имена системных директорий
"Windows" = "c:\\Windows"
"System" = "c:\\Windows\\system"
"Temp" = "c:\\Windows\\Temp"

# Системная переменная PATH указывает, где и в каком порядке
# загружаемые программы  должны искать необходимые для работы
# библиотеки и утилиты.
"Path" = "c:\\Windows;c:\\Windows\\system;y:\\"

# Данная опция указывает, где должны храниться пользовательские
# профили, но в связи с тем, что у нас всего один пользователь,
# она никогда не используется и поэтому закомментирована.
;;"Profile" = "c:\\Windows\\Profiles\\Administrator"

# Какую графическую подсистему нужно использовать для рисования
# экранных объектов.
"GraphicsDriver" = "x11drv"

# Должны ли Windows-программы видеть файлы, у которых первым
# символом имени является точка. Обычно такие файлы считаются
# скрытыми.
;;"ShowDotFiles" = "1"

# Можно ли показывать символические ссылки на директории?
# Обычно это приводит к зависанию Windows-программ,
# пытающихся делать рекурсивный обход директорий. Чаще всего
# в эту ловушку попадаются разнообразные автоматические
# инсталляторы.
;;"ShowDirSymlinks" = "1"

# Указание на скрипт, отвечающий за создание и установку
# правильных иконок для Windows-программ внутри вашего оконного
# менеджера. В качестве менеджеров, к примеру, могут выступать
# GNOME, KDE или CDE.
"ShellLinker" = "wineshelllink"

# Программа, отвечающая за создание и обслуживание меню
# программ оконного менеджера
"LinkProcessor" = "winemenubuilder.exe"

# Тут находятся иконки. Кстати, стоит отметить, что
# для удобства обращения они автоматически конвертируются
# из формата ico в xpm.
"IconsDir" = "c:\\Windows\\Icons"

# Ну а здесь лежит библиотека, отвечающая за работу
# с FreeType-шрифтами.
"FreeTypeLib" = "libcxfreetype.so"


[Restart]
# Программа по идее должна отвечать за перезагрузку Windows.
# Интересно, что такого файла на диске не существует.
# А его функции выполняет скрипт /opt/cxoffice/bin/cxreboot.
# Видимо, кто-то из разработчиков допустил тут ошибку.
"Boot" = "c:\\Windows\\System\\reboot.exe"

# [wineconf]

[Version]
# Какую версию Windows должен имитрировать эмулятор?
# Опция может принимать значения: win95, win98, winme, nt351,
# nt40, win2k, winxp, win2k3, win20, win30, win31. Некоторые
# программы будут работать только под определенной версией.
"Windows" = "win98"

# Версия MS-DOS. Нужна для некоторых старых программ.
;;"DOS" = "6.22"
```

Следующая интересная для нас секция конфигурационного файла управляет порядком загрузки динамических библиотек. Иногда случается так, что Windows-приложение лучше работает с родными библиотеками, чем с теми, что предлагает эмулятор.

```
[DllOverrides]
"ole2" = "native, builtin"
"ole2nls" = "native, builtin"
"*comctl32" = "builtin"
"*ICWCONN1.EXE" = "builtin"
"*IEINFO5.OCX" = "builtin"
```

Процедура загрузки библиотек читает опции слева направо. Соответственно, к примеру, для ole32 ключевое слово native означает, что эмулятор должен сначала найти настоящую Windows-библиотеку и только в случае неудачи пытаться подменить ее встроенной версией. Как видите, описание имен библиотек позволяет использовать шаблоны. Конечно, иногда изменение порядка загрузки тех или иных компонентов помогает оживить те или иные безнадежные Windows-приложения. Но все же стоит отдавать себе отчет, что жонглирование библиотеками может запросто довести систему эмуляции до критических ошибок. Впрочем, особенно пугаться не стоит, как только порядок загрузки библиотек будет восстановлен, все сразу же вернется на круги своя.

Разобравшись с понятием подмены загружаемых библиотек, идем дальше по файлу конфигурации к секции, отвечающей за работу с графическими драйверами. Драйвер X11, используемый в Wine для работы с графикой, по замыслу разработчиков должен заниматься отрисовкой элементов графического интерфейса только внутри окон, предоставляемых системным оконным менеджером, и не более того. А менеджер, в свою очередь, не вторгается на территорию окон, предоставленных Wine, и всего лишь управляет их взаимоотношениями с основной системой.

```
[x11drv]
# Количество цветов, выделяемых из системной палитры для нужд
# Windows-программ. Действует только в том случае, когда
```

```
# графическая подсистема работает в режиме глубины цвета
# 8 бит на один пиксель или 256 цветов.
"AllocSystemColors" = "100"
"PrivateColorMap" = "N"

# Пытаться ли использовать более медленные операции вместо
# быстрых для улучшения качества отрисовки. Лично мне
# показалось, что в большинстве случаев разница не будет заметна.
"PerfectGraphics" = "N"

# Указывает, какую глубину цвета необходимо использовать.
# Имеет смысл только для дисплеев, умеющих работать в режиме
# multi-depth. По умолчанию отключено, так как используется
# крайне редко.
;;"ScreenDepth" = "16"

# Имя дисплея, на который нужно выводить графику.
;;"Display" = ":0.0"

# Данная опция показывает, разрешено ли оконному менеджеру
# управлять окнами Wine
"Managed" = "Y"

# В случае если предыдущая опция активирована, можно ли
# позволить менеджеру рисовать рамки окон и прочие элементы.
# Эти две опции улучшают интеграцию эмулируемого приложения
# с остальными программами, но не все задачи могут нормально
# работать в таком режиме.
; WMFrames = "Y"

# Данный режим выступает конкурентом для двух предыдущих опций
# и позволяет работать Wine так, словно это не просто окно,
# а виртуальный десктоп. Соответственно все Windows-приложения
# будут рисовать свои окна только внутри этого десктопа, так как
# им это больше всего подходит. За управление окнами внутри
# виртуального десктопа отвечает Wine. Такой режим позволяет
# избежать взаимодействия Windows-приложений с системным
# оконным менеджером, поэтому он наиболее совместим с моделью
# рисования, используемой настоящей Windows.
;;"Desktop" = "640x480"

# Нужно ли, чтобы приложения, работающие с DirectDraw, могли
# воспользоваться преимуществами работы с графической подсистемой
# в режиме DGA (XFree86 Direct Graphic Architecture). Обязательно
# убедитесь, что у вас есть доступ к устройству /dev/mem
"UseDGA" = "Y"

# Если с DGA поработать не получилось, то можно попробовать
# заставить DirectX использовать разделяемую память для обмена
# данными с видеоподсистемой. Конечно, это будет работать
# медленнее, чем DGA, но все же быстрее, чем применяемый
# стандартно обмен данными с X11 через сокет.
"UseXShm" = "Y"

# Разрешение использовать для отрисовки режим XVideo
"UseXVidMode" = "Y"

# Улучшенное управление окном, получившим фокус ввода.
"UseTakeFocus" = "Y"

# Опция для включения продвинутого способа контроля за мышью.
# Очень полезна для приложений, использующих DirectX
"DXGrab" = "N"

# Включает двойную буферизацию окна, в котором отрисовывается
# десктоп. Применяется только вместе с опцией Desktop, описанной
# выше. Очень помогает при работе с приложениями, интенсивно
# использующими OpenGL .
"DesktopDoubleBuffered" = "N"

# Номер порта для режима XVideo. Нужно использовать только в
# случае, если у нас несколько портов.
;; "XVideoPort" = "43"

# Опция, указывающая, нужно ли работать в синхронном режиме.
# Обычно используется для отладки проблем с X11.
;;"Synchronous" = "Y"
```

Дальше в этой секции идут настройки шрифтов, используемые для работы Windows-программ. Значения всех параметров по умолчанию срабатывают нормально, поэтому заострять внимание на них мы не будем. При наличии интереса все желающие могут ознакомиться с ними самостоятельно. Далее идет описание привязки виртуальных портов к физическим устройствам.

```
[serialports]
"Com1" = "/dev/ttyS0"
"Com2" = "/dev/ttyS1"
"Com3" = "/dev/ttyS2"
"Com4" = "/dev/modem"
```

Вот тут у нас расписаны опции, необходимые для нормальной работы LPT-порта и принтера, если он, конечно, к нему присоединен. Ну и конечно же, описан стандартный механизм работы с файлом подкачки и фильтрами принтера.

```
[parallelports]
"Lpt1" = "/dev/lp0"

[spooler]
"FILE:" = "tmp.ps"
"LPT1:" = "|lpr"
"LPT2:" = "|gs -sDEVICE=bj200 -sOutputFile=/tmp/fred -q -"
"LPT3:" = "/dev/lp3
```

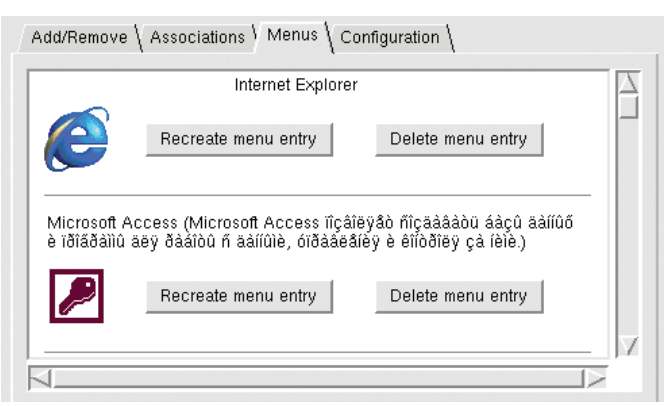
Кстати, стоит отметить, что работа с принтером в CrossOver Office сделана очень удобно. Несмотря на то что в моей системе для печати используется CUPS, приложения Windows автоматически опознали все доступные системе принтеры и без каких-либо дополнительных настроек стали отлично с ними работать.

Следующая сессия указывает, какой стиль иконок, кнопок и прочего оформления использовать для рисования внутри эмулируемых приложений. Доступны стили Win31, Win95 и Win98.

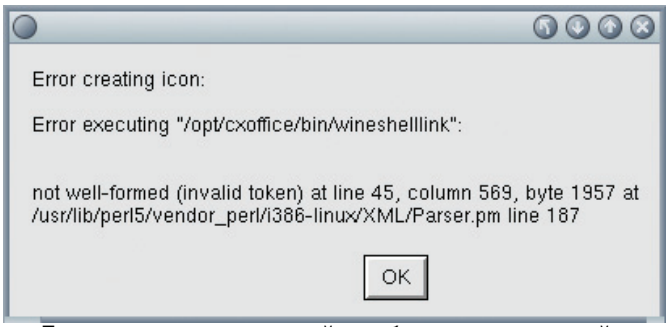
```
[Tweak.Layout]
"WineLook" = "Win95"
```

Далее в конфигурационном файле идут описания настроек звуковой системы, но на них мы тоже не станем останавливаться, потому что они просты и понятны.

Итак, закончив с изучением внутреннего строения CrossOver Office, перейдем к решению насущных проблем. В первой статье мы столкнулись с одним досадным неудобством. При пользовании программой /opt/cxoffice/bin/officesetup было замечено, что на вкладке «Menus» есть список меню, созданных установленными приложениями и выглядящий вот так.



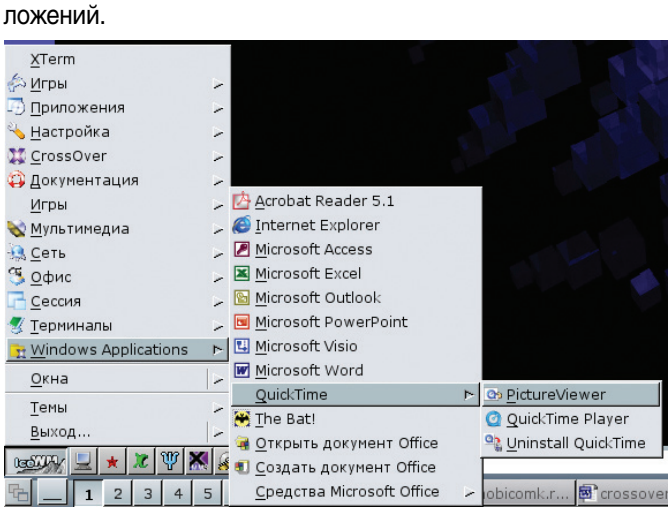
Соответственно с помощью нажатия на кнопку «Recreate menu entry» мы могли бы автоматически создавать пункты меню для отображения в оконном менеджере. Но, к сожалению, с русскими названиями меню CrossOver Office дружить не захотел и при каждой попытке создать то или иное меню выводил вот такую ошибку.



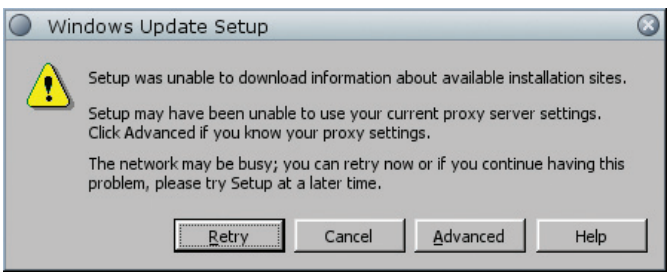
Для исправления данной проблемы, мешающей нам создавать меню, нужно открыть файл $HOME/.gnome2/vfolders/applications.vfolder-info, найти в нем все словосочетания, содержащие следующий набор букв «яПЕДЯРБЮ» и удалить эти странные символы. Такое непонятное слово получается, если неправильно перевести слово «средства», содержащееся в названии меню «Средства Microsoft Office» из кодировки cp1251 в koi8-r. После того как мы выкорчевали все встречные слова «яПЕДЯРБЮ», создание пунктов меню для оконного менеджера должно заработать так, как и положено. После этого конвертируем файл-заготовку в нужную кодировку и обновляем системное меню.

```
$ iconv -f cp-1251 -t utf-8 $HOME/.menu/cxoffice > ↵
    $HOME/.menu/cxoffice
$ update-menus -n -u
```

Подробно о смысле этих действий я писал в предыдущей статье. На следующей картинке в качестве примера вы можете посмотреть, как выглядит мое меню Windows-приложений.

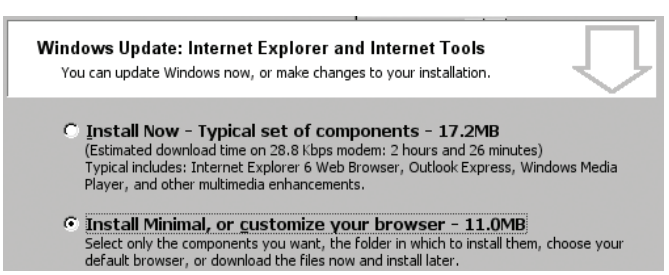


Кстати, это всего лишь малый список Windows-приложений, используемых мною повседневно. Разобравшись с этой проблемой, давайте двинемся дальше. Беда в том, что многие современные приложения для правильного функционирования очень сильно опираются на компоненты, предоставляемые Microsoft Internet Explorer. И все было бы здорово, если бы не вечная гонка за новизной. После инсталляции Microsoft Office 2000 или Microsoft Office XP в системе уже присутствует Microsoft Internet Explorer версии 5.0, но многим устанавливаемым программам этого, к сожалению, недостаточно, им на блюдечке подай шестую версию. Казалось бы, проблема не стоит выеденного яйца, обновить версию вышеуказанного продукта с помощью CrossOver Office не проблема, но не тут-то было. Тем, кто использует для выхода в Интернет прокси-сервер, не повезло, программа инсталляции скачает только первый кусочек обновления и затем выдаст ошибку, сообщающую о невозможности найти сайт, на котором находятся остальные файлы, необходимые для успешного завершения начатого дела.

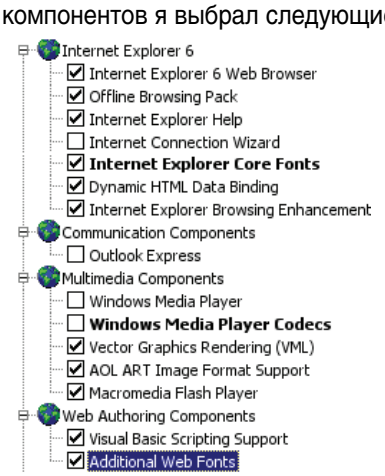


Поэтому нам нужно будет скачать шестую версию вручную и установить ее самостоятельно. Делается это довольно просто.

Для начала убедимся, что версия Windows, под которую маскируется CrossOver, называется win98. Для тех, кто забыл, как это сделать, напоминаю, что нужно посмотреть в файл config и найти в нем секцию [Version]. На отдельной машине или используя VMWare, запускаем операционную систему Windows 98. Данная машина может быть подключена к Интернету через прокси-сервер или напрямую. Скачиваем первую часть инсталлятора, соответствующую языку используемого у вас браузера: http://www.microsoft.com/windows/ie/downloads/critical/ie6sp1/default.asp. После запуска инсталляции используем опцию выборочной установки компонентов.



В дереве компонентов я выбрал следующие пункты:



Думаю, рассказывать об их назначении смысла нет. Если есть желание, вы можете выбрать гораздо больше пунктов, чем сделал я, и поставить себе на машину вдобавок ко всем, mplayer. Впрочем, для цели, ради которой мы все это затеяли, эти добавочные компоненты не критичны.

После того как установка завершится, файлы, скачанные из Сети, будут лежать в директории c:\Windows\Windows Update Setup Files либо c:\Windows\Файлы установки Windows Update. Думаю, все понимают, что название директории зависит от языковой локализации Windows, использованной для скачивания. В директории скорее всего будут лежать файлы с такими именами:

```
ADVAUTH.CAB       FONTCORE.CAB   ie6setup.exe
IELPKAD.CAB       IE_S4.CAB      iesetup.ini
README.CAB        TS95.CAB       BRANDING.CAB
FONTSUP.CAB       IEDOM.CAB      IE_S1.CAB
IE_S5.CAB         MOBILE95.CAB   SCR56EN.CAB
VGX.CAB           CRLUPD.CAB     HELPCONT.CAB
IEEXINST.CAB      IE_S2.CAB      IE_S6.CAB
MPLAYER2.CAB      SETUPW95.CAB   Эту папку можно удалить.txt
filelist.dat      HHUPD.CAB      IE_EXTRA.CAB
IE_S3.CAB         iesetup.dir    OAINST.CAB
SWFLASH.CAB
```

Жизненно необходимы нам только вот эти файлы:

```
IE_S1.CAB         IE_S2.CAB      IE_S3.CAB
IE_S4.CAB         IE_S5.CAB      IE_S6.CAB
IEDOM.CAB         SCR56EN.CAB
```

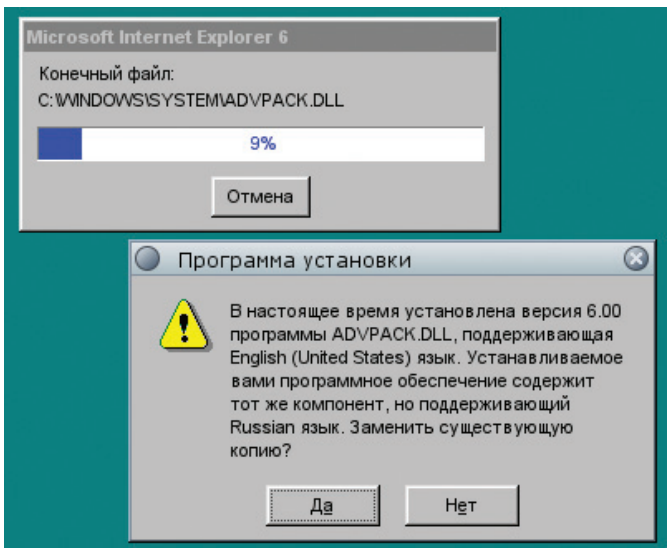
Но все же лучше перестраховаться и на всякий случай скопировать все имеющиеся файлы во временную директорию Windows, которая у нас находится в $HOME/.cxoffice/dotwine/fake_windows/Temp. Распаковываем все файлы с расширением .CAB с помощью утилиты cabextract вот таким образом.

```
#cabextract IE_S1.CAB IE_S2.CAB IE_S3.CAB
```

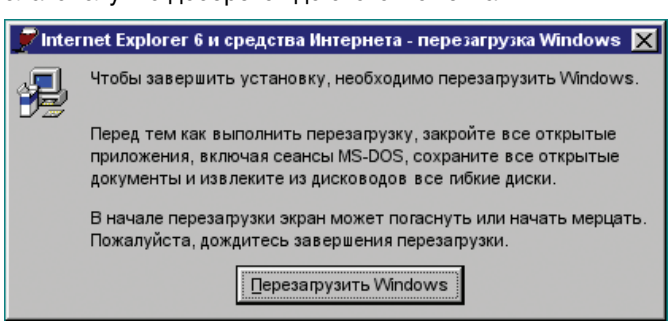
Утилита cabextract существует практически в каждом дистрибутиве Linux. Не стоит пытаться распаковать за один раз больше 3 файлов, иначе будете получать ошибки «Segmentation fault».

Закончив с распаковкой, скопируйте файл nashbase.stf в файл acmsetup.stf и через программу officesetup, как обычно, начинаем инсталляцию Internet Explorer. Выбираем пункт Advanced install и указываем, где у нас находится файл acmsetup.exe. В ответ получаем вот такое весьма приятное для нас изображение.

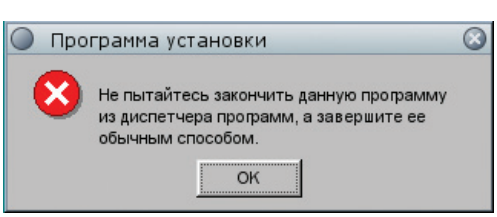
Несколько раз нажимаем кнопки «OK» и «Далее», начинаем любоваться на индикатор установки. Иногда может случиться, что язык устанавливаемых компонентов отличается от того, что было в старых файлах.

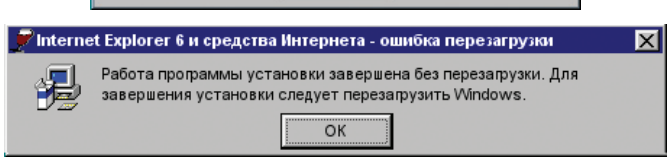


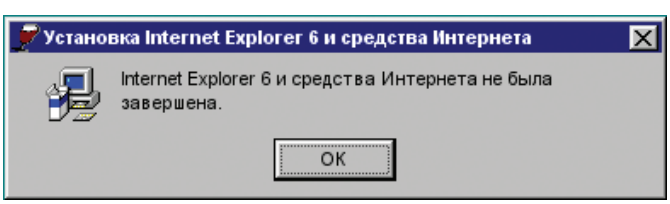
Бояться этого не стоит, просто нажмите кнопку «Да» и на следующий вопрос ответьте «Заменить все такие компоненты». Инсталляция весело побежит дальше и весьма благополучно доберется до этого момента:



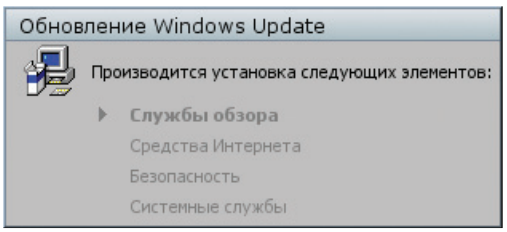
указывающего на то, что все необходимые файлы скопированы и нужно просто перезапустить Windows. Нажимать на единственную доступную на этом диалоговом окне кнопку бесполезно. В ответ будем получать только вот такие ошибки.







И так будет продолжаться бесконечно. Дабы разорвать этот порочный круг, принудительно закрываем окно с работающей инсталляцией. Теперь нужно перезагрузить Windows, делается это с помощью пункта меню Simulate Windows Reboot или вызовом программы /opt/cxoffice/bin/cxreboot. На экране будут наблюдаться следующие явления.



После окончания этого процесса можно будет спокойно работать с обновленным Internet Explorer.

Последним шагом нужно будет установить DCOM98, требующийся для запуска большинства новых программ. Поэтому берем его тут: http://www.microsoft.com/com/dcom/dcom98/download.asp и проводим стандартную инсталляцию.

Я думаю, на сегодня достаточно трудов и мозговых усилий, пора и отдохнуть. Большинство программ будет отлично работать в конфигурации, созданной нашими манипуляциями. В следующей статье мы рассмотрим методы и секреты борьбы с некоторыми особо упорными приложениями.