УДК 681.3.06

В.А. Лавров

НАПИСАНИЕ СВОЕГО MIRROR ВИДЕОДРАЙВЕРА

Существует определенный круг задач, которым необходимо точно детектировать все изменения экрана компьютера. К таким задачам можно отнести системы удаленного доступа к рабочему столу компьютера и программы для создания видеороликов происходящего на экране компьютера. Для этих целей можно воспользоваться технологией компании Microsoft – Mirror Video Driver. В статье рассматриваются различные аспекты написания собственного mirror видеодрайвера.

Системы для удаленного управления рабочим столом известны давно. Первоначально они были созданы для удаленного управления графической средой Unix систем X-Windows. Компания Microsoft также решила включить поддержку удаленного управления в ОС Windows, начиная с версии 2000. Для этого в ядро ОС была добавлена поддержка технологии Mirror Video Driver.

Идея состоит в том, что ОС дублирует все графические операции на mirror видеодрайвер. Таким образом, приложение, использующее mirror драйвер, имеет возможность определять любые изменения экрана. Это позволяет минимизировать объем обрабатываемых данных, по сравнению с простым захватом всего изображения экрана.

В результате минимизации объема данных, происходит минимизация нагрузки на центральный процессор, а значит, большая часть процессорного времени остается на обычные пользовательские приложения.

К сожалению, сама компания Microsoft плохо документировала сведения о mirror видеодрайвере, поэтому в статье раскрываются различные аспекты, которые не отображены в документации и которые в большинстве своем найдены опытным путем.

ЧТО ТРЕБУЕТСЯ ДЛЯ НАПИСАНИЯ ДРАЙВЕРА

Прежде всего, необходимо установить один из вариантов Visual C++. Желательно использовать самые последние версии, например, Visual C++ 7 (Visual Studio .Net 2003).

Кроме этого, для написания драйвера необходим пакет Windows DDK (Driver Development Kit). Всю необходимую информацию о пакете вы сможете найти по ссылке [x]. Минимальная версия пакета должна быть не меньше, чем Windows 2000 DDK.

После установки DDK в меню Программы появится группа Development Kits, в которой находятся ярлыки для настройки переменных окружения для разработки драйвера. Ярлык Check Build предназначен для сборки драйвера в режиме отладки, т.е. он будет скомпилирован без включения оптимизации, и будет включать в себя информацию для отладчика.

Ярлык Free Build предназначен для сборки драйвера для использования конечным пользователем. Компилятор проведет оптимизацию кода и исключит информацию для отладчика из результирующего кода.

Ярлык DDK Documentation открывает документацию компании Microsoft, связанную с разработкой драйверов.

СТРУКТУРА MIRROR ВИДЕОДРАЙВЕРА

Как любой видеодрайвер, mirror драйвер состоит из двух частей – Mirror miniport видеодрайвер и Mirror драйвер дисплея.

Miniport видеодрайвер является специфической программой для каждого видеоадаптера, т.е. пишется под конкретное оборудование. Он работает в нулевом кольце защиты ядра ОС Windows. Для обмена данными с видеоадаптером miniport драйвер использует специальный Video port драйвер. Video port драйвер является независимым от оборудования и поставляется вместе с операционной системой. Miniport драйвер для связи с оборудованием и операционной системой может вызывать только функции Video port драйвера VideoPortXxx.

Таким образом, для написания своего mirror драйвера необходимо создать miniport видеодрайвер и mirror драйвер дисплея.

С ЧЕГО НАЧАТЬ

За основу своего драйвера можно взять пример кода, предложенный компанией Microsoft. В различных версиях DDK пример находится в разных местах, но найти его можно по имени директории, в которой лежат требуемые файлы – mirror. В директории лежат два проекта. В Mirror/dll лежит исходный код mirror драйвера дисплея, а в Mirror/app лежит код win32 приложения уровня пользователя, который умеет запускать драйвер и завершать его работу.

В директории miniport/mirror находится исходный код miniport видеодрайвера. Код содержит самый минимальный набор функций, который должен быть поддержан miniport драйвером. Так как mirror видеодрайвер является не настоящим драйвером оборудования, то нет необходимости реализовывать все функции miniport видео-драйвера. Поэтому, для создания своего драйвера мы просто можем скопировать код mirror miniport драйвера из DDK.

После этого можно считать, что одна часть создаваемого драйвера готова.

ФУНКЦИИ MIRROR ДРАЙВЕРА ДИСПЛЕЯ

Любой драйвер дисплея является динамически загружаемой библиотекой (DLL) и должен поддерживать определенный обязательный набор функций (см. табл.).

Функция	Описание
DrvEnableDriver	Является главной точкой входа в библиотеку
DrvAssertMode	Сбрасывает видеорежим
DrvEnablePDEV	Включает физическое устройство
DrvCompletePDEV	Информирует драйвер о завершении подключения устройства
DrvDisablePDEV	Когда устройство больше не нужно, то освобождаются все ресурсы, выделенные в функции DrvEnablePDEV
DrvEnableSurface	Создает поверхность для специфического устройства
DrvDisableSurface	Информирует драйвер о том, что ранее созданная поверхность больше не нужна

Формальное описание аргументов вышеперечисленных функций имеется в документации к Windows DDK.

Операционная система знает адрес только одной функции в нашей DLL – DrvEnableDriver. Адреса других поддерживаемых функций DLL передает через специальную таблицу, в которой описаны уникальные идентификаторы функций и их адреса в DLL.

Приведем пример описания:

```
static DRVFN gadrvfn[] =
{
 {INDEX_DrvEnablePDEV,(PFN)DrvEnablePDEV},
 {INDEX_DrvCompletePDEV,(PFN)DrvCompletePDEV},
 {INDEX_DrvDisablePDEV,(PFN)DrvDisablePDEV},
 {INDEX_DrvEnableSurface,(PFN)DrvEnableSurface},
 {INDEX_DrvDisableSurface,(PFN)DrvDisableSurface
},
 {INDEX_DrvAssertMode,(PFN)DrvAssertMode},
 …
}
```

И пример передачи этой таблицы операционной системе:

```
BOOL DrvEnableDriver(
  ULONG iEngineVersion,
  ULONG cj,
  PDRVENABLEDATA pded
)
{
  iEngineVersion;
  if (cj >= sizeof(DRVENABLEDATA))
    pded->pdrvfn = gadrvfn;
  if (cj >= (sizeof(ULONG) * 2))
    pded->c = sizeof(gadrvfn) / sizeof(DRVFN);
  if (cj >= sizeof(ULONG))
    pded->iDriverVersion = DDI_DRIVER_VERSION;
    return(TRUE);
}
```

Условия для проверки параметра cj стоят для того, чтобы, даже в случае неправильного вызова, наш драйвер не свалил бы систему, а корректно обработал параметры.

ВНЕШНЯЯ ИНИЦИАЛИЗАЦИЯ MIRROR ДРАЙВЕРА ДИСПЛЕЯ

Загрузка драйвера происходит при запуске операционной системы, а его инициализация производится по требованию.

Для того чтобы активизировать mirror драйвер из win32 приложения уровня пользователя, необходимо выполнить следующую последовательность действий.

Используя API функцию EnumDisplayDevices в цикле, перебираем все видео устройства до тех пор, пока не найдем необходимый Mirror видеодрайвер.

```
DISPLAY_DEVICE dispDevice;
FillMemory(&dispDevice, sizeof(DISPLAY_DEVICE),
0);
dispDevice.cb = sizeof(DISPLAY_DEVICE);
LPSTR deviceName = NULL;
devmode.dmDeviceName[0] = '\0';
INT devNum = 0;
BOOL result;
while (result = EnumDisplayDevices(NULL,
                                   devNum,
                                   &dispDevice,
                                   0))
{
  if (strcmp(&dispDevice.DeviceString[0], driver-
Name) == 0)
    break;
  devNum++;
}
```

Если после выполнения вышеприведенного кода переменная result будет установлена в false, то в системе нет искомого mirror драйвера.

Всем драйверам дисплея назначаются символические имена, которые имеют форму DEVICEn, где n – это номер устройства в ОС.

Если искомый mirror драйвер найден, то его символическое имя находится в поле DeviceKey структуры DISPLAY_DEVICE.

Используя найденный идентификатор, мы можем активизировать mirror драйвер. Для этого необходимо открыть на запись ключ реестра:

HKLM\CurrentControlSet\HardwarePrfiles\Current\System\CurrentControlSet\Services\имя_mirror_драйвера\символическое_имя

И установить значение «Attach.ToDesktop» в 1.

Если значение удалось установить, то активизировать mirror драйвер необходимо функцией ChangeDisplaySettingsEx с последующим вызовом:

```
HDC hdc = CreateDC("DISPLAY", deviceName,
                   NULL, NULL);
```

Для того чтобы после перегрузки ОС Windows не произошла активизация драйвера, необходимо сразу же установить значение «Attach.ToDesktop» в 0.

Для деактивации драйвера мы еще раз вызываем функцию ChangeDisplaySettingsEx, но уже со значением «Attach.ToDesktop» равным нулю.

В этой схеме есть один большой минус – она не работает, если у пользователя нет прав администратора. Обычному пользователю запрещено изменять значения реестра в ключе HKEY_CURRENT_USER.

Для того чтобы преодолеть эту проблему, предлагается в ходе установки драйвера изменить права на необходимую ветку в реестре с тем, чтобы обычные пользователи могли изменять содержимое значения «Attach.ToDesktop».

ВНУТРЕННЯЯ ИНИЦИАЛИЗАЦИЯ ДРАЙВЕРА

Операционная система вызывает функцию драйвера DrvEnablePDEV для внутренней инициализации драйвера.

Внутри этой функции выделяются все необходимые ресурсы. Так как драйвер не может использовать глобальных переменных, то все переменные должны быть привязаны к блоку PDEV указатель, на который передается во все функции драйвера. Таким образом, в любой точке кода драйвера мы будем иметь указатель на область памяти, которая содержит переменные, являющиеся общими для всего кода.

Память под эти переменные должна быть выделена в функции DrvEnablePDEV. Пример выделения памяти под структуру, хранящую общие переменные:

```
DHPDEV DrvEnablePDEV(DEVMODEW *pDevmode,
  PWSTR       pwszLogAddress,
  ULONG       cPatterns,
  HSURF      *ahsurfPatterns,
  ULONG       cjGdiInfo,
  ULONG      *pGdiInfo,
  ULONG       cjDevInfo,
  DEVINFO    *pDevInfo,
  HDEV        hdev,
  PWSTR       pwszDeviceName,
  HANDLE      hDriver
)
{
  PPDEV   ppdev = (PPDEV) NULL;
  ppdev = (PPDEV) EngAllocMem(0, sizeof(PDEV),
ALLOC_TAG);
```

```
  …
  DrvData *drvdata = NULL;
  drvdata = EngAllocMem(FL_ZERO_MEMORY,
sizeof(DrvData), 0x4D495252);
  drvdata->semaphore = NULL;
  drvdata->started = FALSE;
  …
  ppdev->pvTmpBuffer = drvdata;
  return((DHPDEV) ppdev);
}
```

В этом примере структура DrvData содержит все необходимые переменные, и указатель на нее присваивается специальному полю структуры PDEV – pvTmp-Buffer. Это поле предназначено для внутренних нужд драйвера.

После инициализации физического устройства и заполнения структуры PDEV операционная система вызывает драйверную функцию DrvEnableSurface. В этой функции драйвер создает поверхность, на которой должны производиться графические операции.

Для mirror драйвера в качестве поверхности можно использовать область памяти, размера которой хватает для отображения графических данных. Например, если мы установили видеорежим 800×600 при 32 битах на цвет, то размер памяти для поверхности должен быть 800×600×4 = ~1,83 Мб.

Но здесь надо быть очень внимательным. Так как в качестве поверхности для mirror драйвера обычно используется bitmap, то в зависимости от количества бит на цвет размер требуемой памяти рассчитывается по следующей формуле на языке C: ((Ширина × Количество_бит_на_цвет +31) & ~31) / 8 × Высота.

Функция DrvEnableSurface получает на вход указатель на созданную ранее структуру PDEV, поэтому имеется возможность использовать и структуру DrvData. Приведем пример функции:

```
HSURF DrvEnableSurface(
DHPDEV dhpdev)
{
  PPDEV ppdev;
  HSURF hsurf;
  SIZEL sizl;
  ULONG ulBitmapType;
  FLONG flHooks;
  ULONG mirrorsize;
  HBITMAP hbmp;
  DrvData *drvdata;
  ppdev = (PPDEV) dhpdev;
  if(!ppdev->pvTmpBuffer)
    return NULL;
  drvdata = ppdev->pvTmpBuffer;
  sizl.cx = ppdev->cxScreen;
  sizl.cy = ppdev->cyScreen;
  drvdata->cx = ppdev->cxScreen;
  drvdata->cy = ppdev->cyScreen;
  drvdata->bits = ppdev->ulBitCount;
  drvdata->scanlength = ppdev->lDeltaScreen;
  if (ppdev->ulBitCount == 16)
  {
    ulBitmapType = BMF_16BPP;
    flHooks = HOOKS_BMF16BPP;
  }
  else if (ppdev->ulBitCount == 24)
  {
    ulBitmapType = BMF_24BPP;
    flHooks = HOOKS_BMF24BPP;
  }
  else
  {
    ulBitmapType = BMF_32BPP;
    flHooks = HOOKS_BMF32BPP;
  }
  flHooks |= flGlobalHooks;
  mirrorsize = (ULONG)(ppdev->lDeltaScreen *
sizl.cy);
  drvdata->bmpbuffer =
       EngAllocMem(FL_ZERO_MEMORY, mirrorsize,
0x4D495251);
  hbmp =
       EngCreateBitmap(sizl,
       ppdev->lDeltaScreen, ulBitmapType, 0,
       drvdata->bmpbuffer);
  if (hbmp == (HBITMAP) 0)
  {
      return(FALSE);
  }
  if (!EngAssociateSurface((HSURF)hbmp,
       ppdev->hdevEng, flHooks))
  {
     EngDeleteSurface((HSURF)hbmp);
     return(FALSE);
  }
  ppdev->hsurfEng = (HSURF) hbmp;
  return((HSURF)hbmp);
}
```

С помощью функции EngAllolcMem мы выделяем необходимый объем памяти для bitmap. Функция EngCreateBitmap создает дескриптор bitmap, который ассоциируется с поверхностью устройства при помощи функции EngAssociateSurface. В функции используется константа flGlobalHooks. Константа содержит набор флажков, каждый из которых идентифицирует графическую операцию, которую драйвер будет перехватывать у GDI. Например, константа может иметь следующий вид:

```
#define flGlobalHooks   HOOK_BITBLT |
HOOK_TEXTOUT | HOOK_COPYBITS | HOOK_STROKEPATH |
HOOK_FILLPATH | HOOK_LINETO | HOOK_GRADIENTFILL |
HOOK_STRETCHBLT | HOOK_TRANSPARENTBLT |
HOOK_STRETCHBLTROP | HOOK_ALPHABLEND |
HOOK_PLGBLT
```

Каждая константа HOOK_Xxx определяет ту функцию в mirror драйвере дисплея, которая будет вызвана, если такая же функция была вызвана для первичного видеодрайвера. Так драйвер говорит о своем желании перехватывать графические функции.

Все функции EngXxx предоставляются стандартной библиотекой Windows GDI, работающей в ядре ОС.

Если операционной системе ранее созданная поверхность не нужна, то она вызывает функцию драйвера DrvDisableSurface.

ПЕРЕХВАТЫВАЕМЫЕ ГРАФИЧЕСКИЕ ФУНКЦИИ

Любая графическая функция, которая будет перехвачена, должна быть описана в таблице gadrvfn, которая используется в функции DrvEnableDriver.

Приведем пример перехватываемой графической функции, код которой необходимо добавить в mirror драйвер, а указатель на нее поместить в таблицу

gadrvfn. Кроме этого, необходимо выставить соответствующий флаг в константе flGlobalHooks.

```
BOOL DrvCopyBits(
   OUT SURFOBJ *psoDst,
   IN SURFOBJ *psoSrc,
   IN CLIPOBJ *pco,
   IN XLATEOBJ *pxlo,
   IN RECTL *prclDst,
   IN POINTL *pptlSrc
   )
{
  if (psoDst)
  {
    if (psoDst->dhpdev)
    {
      PPDEV ppdev = (PPDEV) psoDst->dhpdev;
      if((ppdev->hsurfEng ==
       psoDst->hsurf)&&(ppdev->pvTmpBuffer))
      {
         DrvData* drvdata =
               (DrvData*)ppdev->pvTmpBuffer;
         EngCopyBits(psoDst, psoSrc, pco, pxlo,
prclDst, pptlSrc);
      }
    }
  }
}
```

Функция DrvCopyBits используется для рисования растровых изображений. Если параметры psoDst и psoSrc представляют собой одну и ту же поверхность, то функция производит перемещение одной части поверхности в другую. Такая операция происходит при скроллировании содержимого окна.

Для активизации функции перехвата DrvCopyBits необходимо в таблицу gadrvfn добавить строчку «{INDEX_DrvCopyBits,(PFN) DrvCopyBits}», а в константу flGlobalHooks флаг HOOK_COPYBITS.

Имеется возможность воспользоваться стандартными средствами GDI для осуществления операций, которые не поддерживаются mirror драйвером дисплея. Как правило, они все начинаются с букв Eng, а дальше идет название перехватываемой функции, а нашем случае – CopyBits.

Для списка всех возможных графических функций для перехвата необходимо смотреть документацию по Windows DDK.

ОТЛАДКА MIRROR ДРАЙВЕРА ДИСПЛЕЯ

Как любой драйвер ядра ОС, mirror драйвер тяжело отлаживать на одном компьютере. Так как драйвер работает в ядре, то любая ошибка может привезти к перегрузке или зависанию компьютера. Для помощи в отладке можно предложить использовать возможность видеодрайвера работать с файлами.

К сожалению, функции драйвера дисплея по работе с файлами очень бедны, поэтому приведем пример функции, для записи текстовых строк в лог файл.

```
void WriteLog(DrvData* drvdata, char* string)
{
  ULONG file;
  char *fileptr;
  int size = strlen(string);
  fileptr =
(char*)EngMapFile(L"\\??\\c:\\bbcap.log",
drvdata->logpos + size, &file);
  if(fileptr)
  {
    memcpy(fileptr + drvdata->logpos, string,
size);
    drvdata->logpos = drvdata->logpos + size;
    EngUnmapFile(file);
  }
}
```

Так как структура DrvData доступна практически в любом месте кода, то в нее добавлено специальное поле logpos, в котором хранится текущая позиция в лог файле.

При помощи функции EngMapFile лог файл отображается на область памяти с заданным размером. Размер задаем с учетом строки, которая должна быть сохранена в лог файле. EngMapFile возвращает указатель на область памяти, в которую файл отображен.

Копируем строку в конец выделенной памяти и закрываем файл используя функцию EngUnmapFile.

Функцию WriteLog можно модифицировать для поддержки сохранения форматированного вывода.

ПЕРЕДАЧА ПАРАМЕТРОВ И КОМАНД ИЗ WIN32 ПРИЛОЖЕНИЯ В MIRROR ДРАЙВЕРА

Для передачи команд из Win32 приложения можно добавить в mirror драйвер дисплея функцию перехватчик DrvDrawEscape.

При вызове win32 API функции DrawEscape GDI передаст управление в mirror драйвер функции перехватчику DrvDrawEscape.

В функции DrvDrawEscape можно напрямую использовать все параметры и указатели, передаваемые функции DrawEscape, так как графическая подсистема работает в контексте того же процесса и потока, в котором была вызвана графическая функция.

Поэтому, имеется возможность передавать данные из Win32 приложения mirror драйверу дисплея и обратно.

ЗАКЛЮЧЕНИЕ

Несмотря на не полную документацию в этой области, написать свой собственный драйвер не сложно.

Автором статьи был разработан mirror видеодрайвер, который используется в коммерческом программном продукте BB FlashBack для определения областей изменений экрана компьютера.

ЛИТЕРАТУРА

1. *Microsoft Corporation.* MSDN Library [Электронный ресурс]. 2004. Режим доступа: http://msdn.microsoft.com/, свободный.
2. *Microsoft Corporation.* Windows Driver Development Kit [Электронный ресурс]. 2004. Режим доступа: http://www.microsoft.com/whdc/devtools/ddk/default.mspx, платный.
3. *David A. Solomon, Mark E. Russinovich.* Inside Microsoft Windows 2000. Third Edition // Microsoft Press.

Статья представлена кафедрой теоретических основ информатики факультета информатики Томского государственного университета, поступила в научную редакцию «Информатика» 30 апреля 2004 г.