ПРАКТИЧЕСКОЕ ЗАНЯТИЕ N2

Программное извлечение флеш-диска

1. Цель работы

Изучение интерфейса USB, используемое в ИС для обмена данными между периферийными устройствами и ЭВМ. Практическое овладение навыками составления программ, работающих с USB-накопителями.

2. Рекомендуемая литература

Агуров П.В. Практика программирования USB. – СПб.: БХВ-Петербург, 2006. с. 69…73, 332…341, 368…374, 566…567.

SetupAPI Reference [Электронный документ]. – Режим доступа: http://msdn.microsoft.com/en-us/library/dd445255.aspx . – 5.11.2009.

GMax. Безопасное извлечение USB-устройств [Электронный документ] / GMax. – Режим доступа: http://wasm.ru/article.php?article=usb_eject . – 5.11.2009.

Программное извлечение USB-диска [Электронный документ]. – Режим доступа: http://superadm.net/index.php?name=pages&op=view&id=126 . – 5.11.2009.

Аблязов Р. Работа с устройствами в Windows [Электронный документ] / Аблязов Р. – Режим доступа: http://pblog.ru/?p=105 . – 5.11.2009.

3. Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.
3.2. Подготовить ответы на контрольные вопросы.
3.3. Подготовить флеш-диск с интерфейсом USB.

4. Контрольные вопросы

4.1. Логическая и физическая архитектура USB?
4.2. Что является инициатором транзакции USB?
4.3. Почему не желательно вынимать USB-накопитель из разъёма без использования безопасного отключения?
4.4. Какой формат имеет PnP-идентификатор USB-устройств?
4.5. Какой формат имеет идентификатор экземпляра устройства USB-накопителя?
4.6. Что делает и какой формат имеет функция SetupDiGetDeviceRegistryProperty?
4.7. Что делает и какой формат имеет функция CM_Get_Device_ID?

4.8. Какие функции из SetupAPI.dll используются для нахождения и отключения устройства?

4.9. Как определить строковый идентификатор производителя и продукта?

5. Задание на выполнение работы

5.1. Используя среду программирования Free Pascal разработать программу, останавливающую **только Вашу флешку** для безопасного извлечения её из разъёма.

Программа может быть выполнена в консольном виде или с графическим пользовательским интерфейсом.

5.2. Подготовить отчёт и отчитаться о проделанной работе преподавателю.

6. Отчёт должен содержать:

6.1 Титульный лист (с названием ВУЗа, кафедры, практического занятия, а также фамилию И.О. студента, подготовившего отчёт).

6.2 Цель работы.

6.3 Графический алгоритм программы с краткими пояснениями.

6.4 Полный листинг программы с комментариями.

7. Общие сведения

При работе с USB-накопителями информации необходимо правильно извлекать эти устройства из системы. В операционных системах семейства Windows для этого имеется функция *«Безопасное извлечение устройства»*, которую можно вызвать командой

rundll32.exe shell32.dll, Control_RunDLL hotplug.dll

Однако данную операцию можно выполнить и программным способом. Существует два метода программного извлечения устройства. Первый метод использует функции библиотеки **SetupAPI.dll**. Если устройство не готово для извлечения в данный момент, то выдаётся соответствующая ошибка или сообщение.

Второй метод использует функции прямого обращения к драйверу. В отличие от первого метода, он может извлечь устройство, даже если оно не готово, но при этом операционная система не уведомляется об отключении диска, что может вызвать ошибку при обращении к диску.

В приложении А приведён листинг программы, которая позволяет безопасно извлекать из системы **первый попавшийся USB-накопитель**. В данной программе проверяется PnP-идентификатор экземпляра устройства. Однако кроме проверки PnP-идентификатора есть и другие

способы нахождения конкретного устройства, например по строке описания этого устройства или идентификатору оборудования.

7.1 Библиотека SetupAPI.dll

SetupAPI это системный компонент, содержащий функции для установки драйвера устройства, и связывающий пользовательское приложение с устройством.

В дальнейшем рассмотрим некоторые функции из двух групп SetupAPI: *Device Installation Functions* и *PnP Configuration Manager Functions*. Первая группа включает функции установки устройств в Microsoft Windows, вторая – дополняет её функциями конфигурирования устройств (CM_xxx).

Функция **SetupDiGetClassDevs** возвращает дескриптор (манипулятор или handle) класса устройства, заданного в качестве параметра. Имеет следующий формат:

hDevInfoSet := **SetupDiGetClassDevs** *(ClassGuid, Enumerator, hwndParent, Flags)*,

где *hDevInfoSet* – имя дескриптора класса;
ClassGuid – идентификатор класса;
Enumerator – системный компонент, определяющий PnP-идентификатор устройства;
hwndParent – дескриптор родительского окна;
Flags – флаг управления функцией. Может принимать пять значений, мы будем использовать флаг по умолчанию DIGCF_DEFAULT = 2.

Жесткие диски и USB-накопители имеют глобальный уникальный идентификатор класса ClassGuid = {4d36e967-e325-11ce-bfc1-08002be10318}.

Функция **SetupDiEnumDeviceInfo** возвращает структуру с информацией об очередном устройстве указанного класса. Если функция вернула значение TRUE, то информация извлечена успешно, а если FALSE, то в большинстве случаев это означает что мы пришли к концу списка.

SetupDiEnumDeviceInfo *(hDeviceInfoSet, MemberIndex, DeviceInfoData)*,

где *hDeviceInfoSet* – дескриптор класса устройств;
MemberIndex – порядковый номер в списке устройств указанного класса;
DeviceInfoData – возвращаемая структура с информацией об устройстве.

Функция **SetupDiGetDeviceRegistryProperty** позволяет получить PnP свойства устройства.

SetupDiGetDeviceRegistryProperty (*hDeviceInfoSet, DeviceInfoData, Property, PropertyRegDataType, PropertyBuffer, PropertyBufferSize, RequiredSize*),

где *hDeviceInfoSet* – дескриптор класса устройств;

DeviceInfoData – указатель на структуру с информацией об устройстве;
Property – параметр, указывающий какое именно свойство требуется получить. Для получения строки с описанием устройства необходимо указать константу или SPDRP_DEVICEDESC (0x00000000) или SPDRP_FRIENDLYNAME (0x0000000C). Для получения идентификатора оборудования (HardwareID) необходимо указать константу SPDRP_HARDWAREID (0x00000001);
PropertyRegDataType – указатель на переменную, в которую помещается тип возвращаемых функцией данных;
PropertyBuffer – указатель на буфер, в который возвращается значение указанного свойства. Если этот параметр указа как null и *PropertyBufferSize* указан как 0, то функция возвращает в RequiredSize необходимый размер буфера;
PropertyBufferSize – размер буфера для получения значения свойства;
RequiredSize – дополнительный параметр для получения размера буфера, если не используется, то null.

Функция **SetupDiDestroyDeviceInfoList** удаляет всю информацию об устройствах указанного класса, и очищаем память. В качестве параметра этой функции указывается дескриптор класса устройств (*hDeviceInfoSet*), который предварительно был получен функцией SetupDiGetClassDevs.

Функция **CM_Get_Parent** получает дескриптор родительской ветки в дереве устройств локальной машины.

CM_Get_Parent *(pdnDevInst, dnDevInst, ulFlags)*,

где *pdnDevInst* – возвращаемый указатель на идентификатор родительского устройства;
dnDevInst – идентификатор устройства;
ulFlags – не используется, должен быть нулём.

Функция **CM_Get_Device_ID_Size** возвращает размер строки идентификатора устройства.

CM_Get_Device_ID_Size (*pulLen, dnDevInst, ulFlags*),

где *pulLen* – указатель на переменную для записи длины строки;
dnDevInst – идентификатор устройства;
ulFlags – не используется, должен быть нулем.

Функция **CM_Get_Device_ID** возвращает текстовый идентификатор экземпляра устройства ID.

CM_Get_Device_ID (*dnDevInst, Buffer, BufferLen, ulFlags*),

где *dnDevInst* – дескриптор устройства;
Buffer – указатель на буфер для записи строки идентификатора устройства;
BufferLen – длина строки идентификатора устройства;
ulFlags – не используется, должен быть нулем.

Функция **CM_Request_Device_Eject** выполняет безопасное извлечение устройства, а если это не возможно, то возвращает информацию об ошибке.

CM_Request_Device_Eject (*dnDevInst, pVetoType, pszVetoName, ulNameLength, ulFlags*),

где *dnDevInst* – дескриптор устройства;
pVetoType – дополнительный параметр для возвращения кода ошибки, если отказано в извлечении устройства;
pszVetoName – дополнительный параметр для возвращения текстового описания ошибки, в случае отказа в извлечении устройства;
ulNameLength – максимальная длина текстового описания ошибки;
ulFlags – не используется, должен быть нулем.

Функция **CM_Locate_DevNode** позволяет получить дескриптор устройства по строке идентификатора.

CM_Locate_DevNode (*pdnDevInst*, *pDeviceID*, *ulFlags*),

pdnDevInst – указатель на возвращаемый дескриптор устройства;
pDeviceID – указатель на строку идентификатора устройства;
ulFlags – флаг управления функцией. Может принимать четыре значения, мы будем использовать CM_LOCATE_DEVNODE_NORMAL = 0.

Функция **CM_Get_DevNode_Status** позволяет получить статус устройства, по которому можно определить, можно ли извлечь данное устройство. Если в статусе (*pulStatus*) возвращается флаг DN_REMOVABLE (0x4000), то устройство можно извлечь.

CM_Get_DevNode_Status (*pulStatus, pulProblemNumber, dnDevInst, ulFlags*),

где *pulStatus* – указатель на переменную со статусом устройства;
pulProblemNumber – указатель на переменную с номером ошибки;
dnDevInst – идентификатор устройства, у которого необходимо проверить статус;
ulFlags – не используется, должен быть нулем.

7.2 PnP-идентификаторы USB-накопителей

Каждое USB-устройство, спроектированное по спецификации PnP, должно иметь идентификатор, который однозначно определяет модель данного устройства.

Идентификатор устройства должен иметь строго определённый для данного класса устройств формат. Для USB-устройства идентификатор имеет следующий формат:

USB\Vid_vvvv&Pid_dddd&Rev_rrrr,

где **vvvv** – код идентификатора производителя, зарегистрированного в Комитете USB-производителей;

dddd – идентификатор, присвоенный производителем данный модели USB-устройства;
rrrr – номер версии разработки.

Все эти поля вводятся как шестнадцатеричные числа.

При идентификации USB-накопителя драйвером порта, последний создаёт новый *идентификатор устройства* (device ID) и набор *идентификаторов оборудования* (hardware ID). Первая строка идентификатора оборудования должна совпадать с идентификатором устройства. Идентификатор оборудования имеет следующий формат

USBSTOR\t*v(8)p(16)r(4)
USBSTOR\t*v(8)p(16)
USBSTOR\t*v(8)
USBSTOR\v(8)p(16)r(1)
v(8)p(16)r(1)
USBSTOR\GenDisk
GenDisk,

где **t*** – тип устройства (для USB-накопителей DISK);
v(8) – идентификатор производителя из 8 символов;
p(16) – идентификатор продукта;
r(4) – идентификатор версии разработки.

Идентификатор оборудования необходим для точного подбора драйвера для устройства.

Кроме *идентификатора устройства* (device ID) каждый накопитель имеет *идентификатор экземпляра* (instance ID), который отличает устройство от других устройств того же типа. Идентификатор экземпляра может определять устройство относительно шины (например, учитывать USB-порт, к которому подключено устройство) или представлять глобально уникальный дескриптор (например, серийный номер устройства). Идентификаторы устройства и экземпляра дают *идентификатор экземпляра устройства* (device instance ID, DIID или код экземпляра устройства), который однозначно идентифицирует экземпляр устройства в системе.

Просмотреть код экземпляра устройства можно в диспетчере устройств, выбрав интересующее дисковое устройство и перейдя на вкладку «Сведений» в свойствах этого устройства.

Например: код экземпляра устройства **USBSTOR\DISK&VEN_JETFLASH&PROD_TS256MJF2A/120&REV_8.07\6&38D7AE47&0&7ZNDZ4S6&0** содержит идентификатор устройства **USBSTOR\DISK&VEN_JETFLASH&PROD_TS256MJF2A/120&REV_8.07** и идентификатор экземпляра **6&38D7AE47&0&7ZNDZ4S6&0**. Идентификатор оборудования для того же устройства содержит:

USBSTOR\DiskJetFlashTS256MJF2A/120__8.07
USBSTOR\DiskJetFlashTS256MJF2A/120__

USBSTOR\DiskJetFlash
USBSTOR\JetFlashTS256MJF2A/120__8
JetFlashTS256MJF2A/120__8
USBSTOR\GenDisk
GenDisk

ПРИЛОЖЕНИЕ А

Листинг 1 – Программа EjectFlesh

```
program EjectFlesh;
{$MODE OBJFPC}

uses
 Windows, strings;

const
 setupapi = 'SetupApi.dll';
 GUID_DEVCLASS_DISKDRIVE: TGUID = (D1: $4D36E967; D2: $E325; D3: $11CE;
D4: ($BF, $C1, $08, $00, $2B, $E1, $03, $18)); // GUID класса накопителей
 GUID_DEVCLASS_USB: TGUID = (D1: $36FC9E60; D2: $C465; D3: $11CF; D4:
($44, $45, $53, $54, $00, $00 , $00, $00)); // GUID класса хост-контроллера и USB
хабов;

type
 HDEVINFO = THandle;

 PSP_DEVINFO_DATA = ^SP_DEVINFO_DATA;
  SP_DEVINFO_DATA = packed record
                      cbSize : DWORD;
                   ClassGuid : TGUID;
                     DevInst : DWORD;
                    Reserved : DWORD;
  end;

var
                    q: char;
      hDevInfoSet: HDEVINFO;
           DevInfo: SP_DEVINFO_DATA;
                    i: Integer;
             Parent: DWORD;
        VetoName: PChar;
 VetoNameString: String;

// функции из SetupApi.dll
function SetupDiGetClassDevsA(ClassGuid: PGUID; Enumerator: PChar; hwndParent:
HWND; Flags: DWORD): HDEVINFO; stdcall; external setupapi;
function SetupDiEnumDeviceInfo(DeviceInfoSet: HDEVINFO; MemberIndex: DWORD;
DeviceInfoData: PSP_DEVINFO_DATA): boolean; stdcall; external setupapi;
function SetupDiDestroyDeviceInfoList(DeviceInfoSet: HDEVINFO): boolean; stdcall;
external setupapi;
function CM_Get_Parent(pdnDevInst: PDWORD; dnDevInst: DWORD; ulFlags:
DWORD): DWORD; stdcall; external setupapi;
function CM_Get_Device_ID_Size(pulLen: PDWORD; dnDevInst: DWORD; ulFlags:
DWORD): DWORD; stdcall; external setupapi;
```

```
function CM_Get_Device_IDA(dnDevInst: DWORD; Buffer: PChar; BufferLen:
DWORD; ulFlags: DWORD): DWORD; stdcall; external setupapi;
function CM_Locate_DevNodeA(pdnDevInst: PDWORD; pDeviceID: PChar; ulFlags:
DWORD): DWORD; stdcall; external setupapi;
function CM_Request_Device_EjectA(dnDevInst: DWORD; pVetoType: Pointer;
pszVetoName: PChar; ulNameLength: DWORD; ulFlags: DWORD): DWORD; stdcall;
external setupapi;

function CompareMem(p1, p2: Pointer; len: DWORD): boolean;
var
     i: DWORD;
begin
 result := false;
 if len = 0 then exit;
 for i := 0 to len-1 do
  if PByte(DWORD(p1) + i)^ <> PByte(DWORD(p2) + i)^ then exit;
 result := true;
end;

function IsUSBDevice(DevInst: DWORD): boolean;
var
   IDLen: DWORD;
        ID: PChar;
 IDString: String;
begin
 result := false;
 IDString := '';
 if (CM_Get_Device_ID_Size(@IDLen, DevInst, 0) <> 0) or (IDLen = 0) then exit;
 inc(IDLen);
 ID := GetMemory(IDLen);
 if ID = nil then exit;
 if ((CM_Get_Device_IDA(DevInst, ID, IDLen, 0) <> 0) or (not (CompareMem(ID,
PChar('USBSTOR'), 7)))) then
  begin
   IDString := StrPas(ID);
   FreeMemory(ID);
   exit;
  end;
 IDString := StrPas(ID);
 Write('Eject flash-disk?(y - yes; Any other key - no)');
 ReadLn(q);
 if q = 'y' then result := true;
 FreeMemory(ID);
end;

BEGIN
 DevInfo.cbSize := sizeof(SP_DEVINFO_DATA);
 hDevInfoSet := SetupDiGetClassDevsA(@GUID_DEVCLASS_DISKDRIVE, nil, 0, 2);
 if hDevInfoSet = INVALID_HANDLE_VALUE then exit;
 i := 0;
```

```
    while (SetupDiEnumDeviceInfo(hDevInfoSet, i, @DevInfo)) do
 begin
  if IsUSBDevice(DevInfo.DevInst) then
   begin
    if CM_Get_Parent(@Parent, DevInfo.DevInst, 0) = 0 then
     begin
      VetoName := GetMemory(260);
      if (CM_Request_Device_EjectA(Parent, nil, VetoName, 260, 0) <> 0) then
       begin
        if (CM_Locate_DevNodeA(@Parent, VetoName, 0) <> 0) then
         begin
          FreeMemory(VetoName);
          continue;
         end;
        FreeMemory(VetoName);
        if (CM_Request_Device_EjectA(Parent, nil, nil, 0, 0) <> 0) then continue;
       end;
      VetoNameString := StrPas(VetoName);
      FreeMemory(VetoName);
      break;
     end;
   end;
  inc(i);
 end;
 SetupDiDestroyDeviceInfoList(hDevInfoSet);
END.
```