Программа сбора данных от нескольких клавиатурных устройств для учета рабочего времени

или

Автоматический сбор данных с персональных карт доступа со считывателей класса «Клавиатуры» для учета рабочего времени

by **Emery Emerald**
http://emery-emerald.narod.ru
emerald@mail.ru

Предлагается решение и программа для автоматической фиксации входа / выхода сотрудников через компьютеризированную проходную с целью сбора данных для учета рабочего времени.

Февраль 2011 года

Прилагаемые файлы:
ElPath.zip (http://emery-emerald.narod.ru/Others/ElPath.000 - измените расширение в zip)
ElPath.pdf (http://emery-emerald.narod.ru/Others/ElPath.pdf)

Содержание

Введение

Появились новые средства автоматизации учета рабочего времени, которые представляют интерес для предприятий ищущих недорогие простые решения без особых изысков. Одним из таких вариантов может быть применение китайских бесконтактных считывателей типа **CR-10E** с **USB** интерфейсом. (рис. 1)

Рис. 1. Китайский 125KHz Proximity Card Reader CR-10E с USB интерфейсом.

Есть и другие похожие считыватели **CR-10**M для бесконтактных карт с чипом на **13.56MHz**, но на нашем предприятии уже используются килогерцовые пропуска, поэтому мы не будем останавливаться на мегагерцовых. По логике вещей, наша программа не должна от этого зависеть.

Эти считыватели хороши тем, что, во-первых, не требуют установки специальных драйверов, поскольку относятся к стандартному классу устройств «Клавиатуры», а, во-вторых, они непосредственно могут быть подключены к компьютеру, например к очень дешевому нетбуку. Конечно, есть резон использовать их в относительно теплых помещениях, чтобы не повредить электронику. В общем, место вахтера или охранника вполне подойдет. Нетбук и считыватели можно повесить на стену, одни из которых будет фиксировать вход сотрудников, а другой выход. К нетбуку можно протянуть сетевой кабель от сервера, который будет обрабатывать данные со всех устройств контроля рабочего времени персонала, где и будет производиться окончательная обработка информации собранной считывателями. Примерный прототип показан на рис. 2.

Рис. 2. Примерный прототип комплекта по учету рабочего времени сотрудников.

Конечно, реальный образец будет висеть на стене, в каком-нибудь железном ящике, чтобы ограничить доступ посторонним. Но может быть непосредственно на столе вахтера или охранника, это уже на ваше усмотрение :) .

1. Технология Raw Input

Технология Raw Input не является широко известной. Тем не менее, медленно, но верно народ осваивает ее. Но сначала пару слов, почему в ней возникла необходимость.

Известно, что, операционные системы Майкрософт все сообщения от однотипных устройств объединяют в общий поток данных. Т.е. несколько клавиатур или мышей объединяются в одну системную клавиатуру или одну системную мышь. В нашем случае, считыватели распознаются операционной системной **Windows** как клавиатуры. Так, подсоединив два считывателя **EM-Marin USB CR-10E** к одному компьютеру (или дешевому нетбуку), один из них должен распознаваться как вход, а другой, как выход. Особенно это актуально, для многосменного учета, неполных отметок (только вход или только выход) и работе вне графика, когда не всегда можно явно вычислить зашел человек на предприятие или вышел.

В итоге у нашего компьютера будет три устройства класса **«Клавиатуры»**. Они принадлежат более общему семейству **HID** устройств, в терминологии **Майкрософт** (**Human Input Devices**, в свободном переводе, **устройства ввода данных человеком**).

Системная клавиатура также должна отличаться программно от считывателей. Осуществить это можно многими разными способами. Например, используя собственный драйвер клавиатуры, или перехват сообщений устройств на уровне ядра **Windows**, до того как операционная система объединит однородные потоки. В принципе, работа с **HID** устройствами осуществлялась до сих пор как работа с сокетами (аналогия) либо **Direct Input** (**DirectX**), причем последний более ограничительный в использовании. Есть и другие более экзотически варианты. Например, осуществив аппаратную «обёртку» считывателя, можно получить, скажем, в сетевом пакете, данные об этом устройстве. Некоторые производители оборудования идут по этому пути, но, как нам кажется, в этом нет особой необходимости.

Однако самый простой способ это использование технологии **Raw Input**, которая появилась начиная с **WinXP клиент** и **Win2003 сервер**.

Чтобы получить общее представление об этой технологии можно скачать файлы примеров по адресу http://www.codeproject.com/KB/system/rawinput.aspx (для этого достаточно бесплатно зарегистрироваться на этом сайте). Других подходящих примеров, различающих ввод от несколько клавиатур я не нашел в Интернете, хотя частичной информации достаточно много.

Мне не понравилось в этом примере то, что он использует **C#** и **.Net**, а не **C++** и **WinApi**. Поэтому я решил написать код под **VC++ v.6.0, sp.6**, без использования **MFC**. Для этого нужно использовать специальные функции **Raw Input** из **user32.dll** посредством **LoadLibrary / GetProcAddress**. Кроме того, нужно еще найти в Интернете описания соответствующих структур, используемых в этой технологии (если не использовать **Net Framework**). К счастью, такой файл был обнаружен в свободном доступе, это **Rawinput.h** (http://files.codes-sources.com/fichier.aspx?id=48710&f=rawinput.h&lang=en). Поэтому мы просто подключили его к нашему проекту.

Собственный код программиста, использующего **Raw Input**, будет заключаться в регистрации обработчиком его программного окна всех устройств класса **«Клавиатура»** и обработке сообщений **WM_INPUT**, поступающих от этих устройств. Вся прелесть этого сообщения в том, что помимо собственно данных **WM_INPUT** шлет нам также хэндл устройства, которое отправляет эти данные. Тем самым все клавиатуры (или, если необходимо, все мыши или другие однотипные **HID**-устройства) будут иметь различные хэндлы своего оборудования. По хэндлу можно вычислить и всю другую информацию, относящуюся к определенному устройству (номер порта, подтип класса устройств, GUID, может быть серийный номер и т.д. и т.п.), т.е. ту информацию, что регистрируется в реестре при инициализации данного устройства.

Нетбуки хороши тем, что относительно недороги, легко подключаются в сеть, с выходом на сервер. При отключении питания, батареи нетбука хватит на работу до 8 часов. А данные считыватели с **USB** интерфейсом привлекательны тем, что легко программируются самостоятельно, что мы и демонстрируем в данной статье. Так что молодцы китайцы, что производят такое недорогое оборудование и те фирмы, которые продают его на нашей территории :). Кроме того, оно поддерживает стандартные карточки доступа типа **El Marin Proximity Card**, используемых и на других системах контроля доступа и учета рабочего времени.

Также следует отметить дополнительное удобство китайских считывателей, они помимо внутреннего номера карты доступа (отличающегося от внешнего номера, напечатанного на самой карточке) генерят еще код возврата (**0x0D = 13**), что позволяет задействовать стандартную технику обработки данных на диалогах **WinApi** (обрабатывая код доступа не посимвольно, а пакетно).

Однако, похоже, китайские считыватели не дают нам своего серийного номера. Этой информации нет ни в реестре, ни в структурах поставляемых Raw Input. Видимо микросхемы просто скопированы без внутренней идентификации. Более-менее уникальная информация это то, что мы здесь назвали портом (**Port**). Она действительно зависит от **USB**-порта, одинаковая на одинаковых портах и разная на разных.

Хэндл считывателя также отчасти привязан к порту, но при перестыковке шнура меняет свое значение, хотя может и вернуться к исходному. Поэтому мы сохраняем всю эту избыточную информацию, в том числе и определение того, как инициализирован данный считыватель: на вход или на выход. Избыточная информация поможет отслеживать некоторые внештатные ситуации и повышает надежность получаемой информации.

Собственно код обработчика главного окна показан ниже.

```
//////////////////////////////////////////////////////////////////////
// DialogProc - Оконная процедура основного диалогового окна
//////////////////////////////////////////////////////////////////////
BOOL CALLBACK  MainBoxProc(HWND hDlg, UINT nMsg, WPARAM wParam, LPARAM lParam) {
  switch(nMsg) {
    case WM_INITDIALOG:
      return HANDLE_WM_INITDIALOG(hDlg, wParam, lParam, MainBoxOnInitDialog);
    case WM_COMMAND:
      return HANDLE_WM_COMMAND(hDlg, wParam, lParam, MainBoxOnCommand);
    case WM_SYSCOMMAND:
      return HANDLE_WM_SYSCOMMAND(hDlg, wParam, lParam, OnSysCommand);
    case WM_INPUT:
      OnInput(hDlg, (HRAWINPUT) lParam, TRUE);
      return FALSE;
    case WM_TIMER:
      return HANDLE_WM_TIMER(hDlg, wParam, lParam, OnTimer);
    case WM_CLOSE:
      return HANDLE_WM_CLOSE(hDlg, wParam, lParam, OnClose);
    default:
      return FALSE;
  }
}  // DialogProc
```

Инициализация **Raw Input** происходит в функции **MainBoxOnInitDialog**.

```
//////////////////////////////////////////////////////////////////////
// MainBoxOnInitDialog - Функция обработки сообщений об инициализации диалога
//////////////////////////////////////////////////////////////////////
BOOL MainBoxOnInitDialog(HWND hDlg, HWND wParam, LPARAM lParam) {
  if(!hDlg) {
    MessageBox(NULL, "MainBoxOnInitDialog: hDlg = NULL!", "Error", MB_OK);
    return FALSE;
  }

  if(!CreateDbfFile(g_acDbfFile)) {
    //MessageBox(NULL, "MainBoxOnInitDialog: CreateDbfFile is wrong!", "Error", MB_OK);
    return FALSE;
  }

  if(!GetIniFile(g_acIniFile, TRUE))
     MessageBox(NULL, "MainBoxOnInitDialog: GetIniFile is wrong!", "Error", MB_OK);

  g_bButCheck = IsDlgButtonChecked(hDlg, IDC_CHECK_NEWEDIT);  // Отметка ручного редактирования

  // Получить хэндлы контролов диалога
  if(!GetHandlesOfDlgCtrls(hDlg))
     return FALSE;

  // Получить адреса функций Raw Input из user32.dll
  if(!GetApiAddr())
     return FALSE;

  // Установить иконку
  SendMessageA(hDlg, WM_SETICON, (WPARAM) 1, (LPARAM) LoadIconA(g_hInstance, MAKEINTRESOURCE(IDI_SMALL)));

  RAWINPUTDEVICE RID = {0};
  RID.usUsagePage = 0x01;
  RID.usUsage = 0x06;  // HID Keyboards
  RID.dwFlags = RIDEV_INPUTSINK;
```

```
  RID.hwndTarget = hDlg;

  //*** Регистрация Raw Input
  if(!RegisterRawInputDevices(&RID, 1, sizeof(RAWINPUTDEVICE))) {
    MessageBox(NULL, "MainBoxOnInitDialog: RegisterRawInputDevices() is wrong!", "Error", MB_OK);
    return FALSE;
  }

  SetTimer(hDlg, IDT_TIMER, 1000, NULL);  // Установить таймер с интервалом в 1 сек

  return TRUE;
}  // MainBoxOnInitDialog
```

А вот сама функция **OnInput**.

```
////////////////////////////////////////////////////////////////////////
// OnInput - Функция обработки сообщения WM_INPUT
////////////////////////////////////////////////////////////////////////
void OnInput(HWND hDlg, HRAWINPUT hRawInput, BOOL bIsMainBox) {
  // При ручном редактировании данных обработка сообщения WM_INPUT не нужна
  if(bIsMainBox && g_bButCheck)
     return;

  if(!hDlg) {
    MessageBox(NULL, "OnInput: hDlg is NULL!", "Error", MB_OK);
    return;
  }

  // Размер буфера для Raw Input данных
  UINT dwSize = 0;

  // Определяем размер dwSize для Raw Input данных
  GetRawInputData(hRawInput, RID_INPUT, NULL, &dwSize, sizeof(RAWINPUTHEADER));

  // Выделяем буфер размера dwSize
  LPBYTE acBuf = new BYTE[dwSize];

  if(!acBuf) {
    MessageBox(NULL, "OnInput: acBuf is NULL!", "Error", MB_OK);
    return;
  }

  // Получаем Raw Input данные
  if(GetRawInputData(hRawInput, RID_INPUT, acBuf, &dwSize, sizeof(RAWINPUTHEADER)) != dwSize) {
    MessageBox(NULL, "OnInput: GetRawInputData() is wrong!", "Error", MB_OK);
    return;
  }

  RAWINPUT *pRawInp = (RAWINPUT *) acBuf;

  if(pRawInp->header.dwType == RIM_TYPEKEYBOARD) {  // KEYBOARD
    HANDLE hDevice = pRawInp->header.hDevice;

    //wsprintf(g_acStr, "%08X", hDevice);
    wsprintf(g_acStr, "%x", hDevice);

    if(bIsMainBox) {
      if(!SetDlgItemText(hDlg, IDC_EDIT_NEWHANDLE,  g_acStr)) {
        MessageBox(NULL, "OnInput: SetDlgItemText() for IDC_EDIT_NEWHANDLE is wrong!", "Error", MB_OK);
        return;
      }
    }  else {
      if(g_bInRadio)
         wsprintf(g_acInHandle, "%s", g_acStr);  // Хэндл устройства входа
      else
         wsprintf(g_acOutHandle, "%s", g_acStr);  // Хэндл устройства выхода
      /*
      if(g_bInRadio) {
        if(!SetDlgItemText(hDlg, IDC_EDIT_INHANDLE,  g_acStr)) {
```

```
    MessageBox(NULL, "OnInput: SetDlgItemText() for IDC_EDIT_INHANDLE is wrong!", "Error", MB_OK);
    return;
   }
  } else {
   if(!SetDlgItemText(hDlg, IDC_EDIT_OUTHANDLE,  g_acStr)) {
    MessageBox(NULL, "OnInput: SetDlgItemText() for IDC_EDIT_OUTHANDLE is wrong!", "Error", MB_OK);
    return;
   }
  }
  */
}

UINT nSize = 0;

// Первый вызов
if(GetRawInputDeviceInfo(hDevice, RIDI_DEVICENAME, NULL, &nSize) < 0) {
  MessageBox(NULL, "OnInput: First GetRawInputDeviceInfo() is wrong!", "Error", MB_OK);
  return;
}

TCHAR *acDevName = new TCHAR[nSize];

if(!acDevName) {
  MessageBox(NULL, "OnInput: acDevName is NULL!", "Error", MB_OK);
  return;
}

// Второй вызов
if(GetRawInputDeviceInfo(hDevice, RIDI_DEVICENAME, acDevName, &nSize) < 0) {
  MessageBox(NULL, "OnInput: Second GetRawInputDeviceInfo() is wrong!", "Error", MB_OK);
  return;
}

//*** Поиск строки - индентификатора для всех устройств класса "Клавиатуры"

// Подстрока, которой оканчивается нужный нам идентификатор устройства класса "Клавиатура"
//TCHAR acStrEnd[] = "&0&0000#{";
TCHAR acStrEnd[] = "&0";

TCHAR *pDest = strstr(acDevName, acStrEnd);

if(!pDest) {
  wsprintf(g_acStr, "OnInput: acDevName = '%s'", acDevName);
  MessageBox(NULL, g_acStr, "Error", MB_OK);

  return;
}

UINT nResult = (UINT) (pDest - acDevName);

acDevName[nResult] = NULL;

int cChar = '&';  // Последний символ, после которого начинается нужный нам идентификатор

pDest = strrchr(acDevName, cChar);

if(!pDest) {
  MessageBox(NULL, "OnInput: pDest is NULL!", "Error", MB_OK);
  return;
}

nResult = (UINT) (pDest - acDevName + 1);

wsprintf(g_acStr, "%s", &acDevName[nResult]);

if(bIsMainBox) {
  if(!SetDlgItemText(hDlg, IDC_EDIT_NEWPORT,  g_acStr)) {
    MessageBox(NULL, "OnInput: SetDlgItemText() for IDC_EDIT_NEWPort is wrong!", "Error", MB_OK);
    return;
```

```
    }

    if(!strcmp(g_acStr, g_acInPort)) {  // Устройство входа
      if(!SetDlgItemText(hDlg, IDC_EDIT_NEWIO,  g_acI)) {
        MessageBox(NULL, "OnInput: SetDlgItemText(g_acI) for IDC_EDIT_NEWIO is wrong!", "Error", MB_OK);
        return;
      }

      if(!SetDlgItemText(hDlg, IDC_STATIC_NEWOPER,  g_acInput)) {
        MessageBox(NULL, "OnInput: SetDlgItemText(g_acInput) for IDC_STATIC_NEWOPER is wrong!", "Error", MB_OK);
        return;
      }
    } else {
      if(!strcmp(g_acStr, g_acOutPort)) {   // Устройство выхода
        if(!SetDlgItemText(hDlg, IDC_EDIT_NEWIO,  g_acO)) {
          MessageBox(NULL, "OnInput: SetDlgItemText(g_acO) for IDC_EDIT_NEWIO is wrong!", "Error", MB_OK);
          return;
        }

        if(!SetDlgItemText(hDlg, IDC_STATIC_NEWOPER,  g_acOutput)) {
          MessageBox(NULL, "OnInput: SetDlgItemText(g_acOutput) for IDC_STATIC_NEWOPER is wrong!", "Error", MB_OK);
          return;
        }
      } else {  // Неопределенное устройство
        if(!SetDlgItemText(hDlg, IDC_EDIT_NEWIO,  g_acU)) {
          MessageBox(NULL, "OnInput: SetDlgItemText(g_acU) for IDC_EDIT_NEWIO is wrong!", "Error", MB_OK);
          return;
        }

        if(!SetDlgItemText(hDlg, IDC_STATIC_NEWOPER,  g_acUndef)) {
          MessageBox(NULL, "OnInput: SetDlgItemText(g_acUndef) for IDC_STATIC_NEWOPER is wrong!", "Error", MB_OK);
          return;
        }
      }
    }
  } else {
    if(g_blnRadio)
      wsprintf(g_acInPort, "%s", g_acStr);  // Идентификатор устройства входа
    else
      wsprintf(g_acOutPort, "%s", g_acStr);  // Идентификатор устройства выхода
    /*
    if(g_blnRadio) {
      if(!SetDlgItemText(hDlg, IDC_EDIT_INPort,  g_acStr)) {
        MessageBox(NULL, "OnInput: SetDlgItemText() for IDC_EDIT_INPort is wrong!", "Error", MB_OK);
        return;
      }
    } else {
      if(!SetDlgItemText(hDlg, IDC_EDIT_OUTPort,  g_acStr)) {
        MessageBox(NULL, "OnInput: SetDlgItemText() for IDC_EDIT_OUTPort is wrong!", "Error", MB_OK);
        return;
      }
    }
    */
  }
 }
}  // OnInput
```

Другие детали вы можете посмотреть в прилагаемом коде.

Работа главного окна продемонстрирована на рис. 3.

2. Инициализация устройств на вход и выход

Все это, однако, хорошо, но нам еще желательно знать, какое именно устройство зарегистрировало вход, а какое выход. Понятно, что направление доступа мы можем задать самостоятельно. Проще всего это сделать в отдельном диалоге вроде (рис. 4)

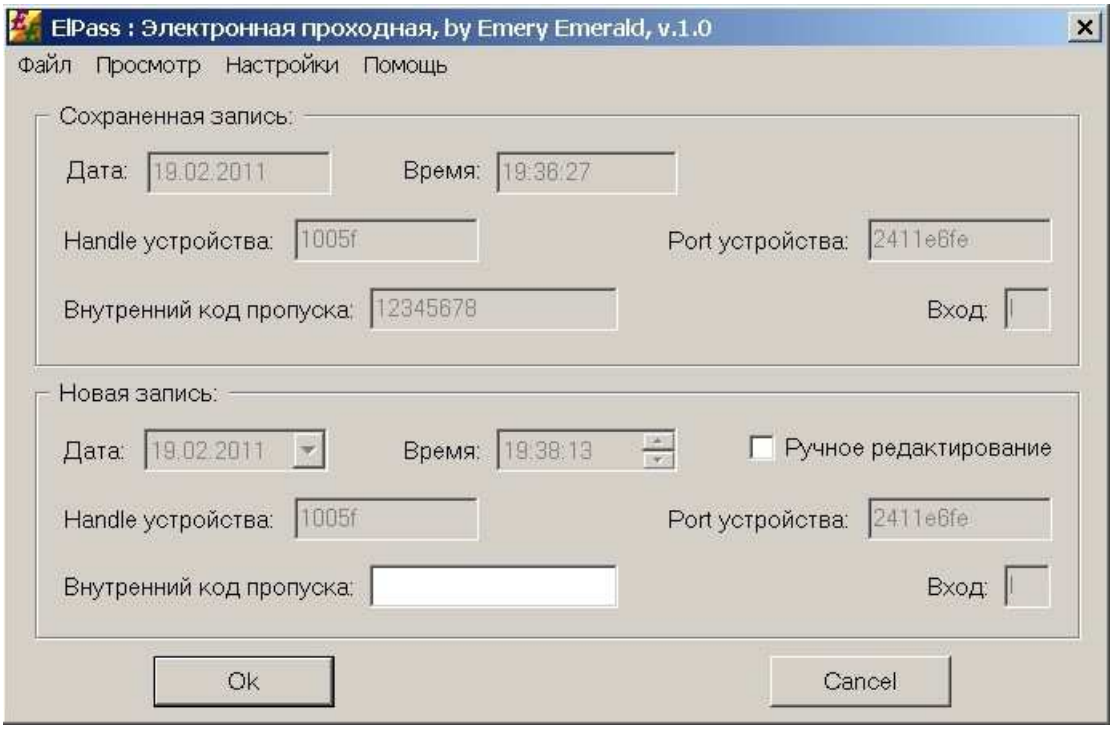

Рис. 3. Работа главного окна электронной проходной ElPath.

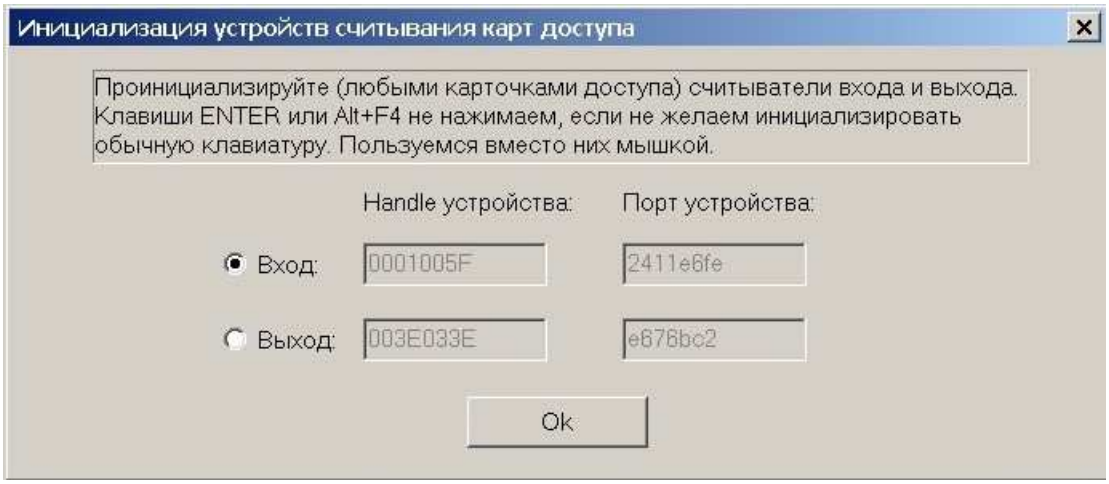

Рис. 4. Инициализация устройств на выход и выход.

Вход и выход мы обозначаем соответственно буквами **«I»** (**Input**) и **«O»** (**Output**). Другие устройства, например обычная клавиатура у нас обозначаются **«U»** (**Undefined**).

Данные инициализации файлов сохраняются в файле **devices.ini**, который должен находится рядом с **ElPath.exe**. Естественно, для разных компьютеров и устройств он будет разный.

3. Формат сохранения данных

Сохранять данные можно уже многими разными способами, начиная от текстового формата и заканчивая любым форматом базы данных. Мы выбрали второй по распространенности (после текстового) формат базы данных dbf (поддерживаемого, в том числе, **Visual FoxPro**). Самостоятельную работу с dbf-файлами мы уже продемонстрировали в своей статье *«Создание и проецирование в память существующих DBF файлов как альтернатива сериализации данных при работе с модифицированными списками CListCtrl в виртуальном режиме на диалогах MDI приложения»* (http://emery-emerald.narod.ru/Cpp/2E14.html). Только в данном случае **MFC** мы не использовали. В нашем случае ситуация даже проще. Файл у нас один единственный, фиксированной структуры в котором только осуществляется последовательная допись принимаемых считывателями данных. Можно, в принципе, генерировать ежедневные (или другой периодичности) файлы, привязанные к дате. К нашему файлу

базы данных **ElPath.dbf** можно подключиться, даже во время его работы, например средствами **Visual FoxPro**. Вот простой пример кода на **VFP**, копирующие все данные из открытого файла.

```
*************************************************************************
CLEAR
SET TALK OFF
SET SAFETY OFF

* SELECT 0
USE ElPath SHARED

SELECT * FROM ElPath INTO TABLE Temp;

CLOSE ALL
QUIT
*************************************************************************
```

Для сохранения корректного состояния dbf-файла в любой момент времени, мы каждую новую запись завершаем признаком конца dbf-файла (байт 0x1A) и изменяем счетчик числа записей файла в его заголовке непосредственно после каждого сохранения текущей записи. Затем для новой записи смещаемся на один байт назад, чтобы перезаписать конечный байт и повторяем процедуру заново. Может быть, это не идеальное решение, так как головка диска делающая постоянную допись в конец файла вынуждена перемещаться в начало файла, чтобы изменить текущий счетчик записей. Если же нам нет необходимости получать доступ к открытому файлу, генерируемому в определенном промежутке времени, то тогда счетчик записей в заголовке файла можно отложить на момент закрытия файла при выходе из программы. Заметим, что при повторном запуске программы, файл **ElPath.dbf** не создается заново, а открывается на дозапись.

Так что мы не будем много останавливаться на этом вопросе, всю необходимую информацию вы можете найти сами.

4. Просмотр данных

В этой версии пока еще не реализован собственный просмотр данных, но в следующих планируется просмотр получаемого dbf-файла в самой программе, по аналогии как это уже сделано в цитируемой выше статье, хотя это и не особо актуально, так как dbf-файл можно просмотреть многими внешними программами просмотрщиками. Например, с помощью плагинов файловой оболочки **Total Commander**, либо в самом **Visual FoxPro**. Важнее наличие самого файла данных, работать с которым можно уже в существующих базах данных, с имеющимися провайдерами данных либо независимо.

5. Практическая реализация

Наша первая проходная уже оборудована парой считывателей и нетбуком, подключенным к сети. Пока все работает прекрасно (рис. 5-7), несмотря на «колхозный» вид 🤓. К нетбуку имеет полный доступ только администратор системы, т.е. я, для остальных будет выделяться ограниченный доступ по мере необходимости. Со своего сервера я могу даже удаленно перезагружать операционную систему нетбука с последующим автоматическим восстановлением соединения.

6. Дальнейшее использование

Эта программа позволяет только осуществлять сбор данных от считывателей с карт доступа. Чтобы получать уже более интересную информацию по контролю и учету рабочего времени, нужно уже подключаться к таблицам баз данных содержащих информацию о сотрудниках, их картах доступа, графиках работы и т.д. и т.п. В следующей программе мы планируем вести контроль доступа, а в третьей уже учет доступа (рабочего времени) с автоматическим построением электронных табелей сотрудников для использования их расчете заработной платы. Возможна также интеграция данного учета и контроля рабочего времени в нашу собственную конфигурацию 1С77 **«Учет ресурсов и начислений»**, хотя это не обязательное для всех решение.

Рис. 5. Нетбук в защитном корпусе.

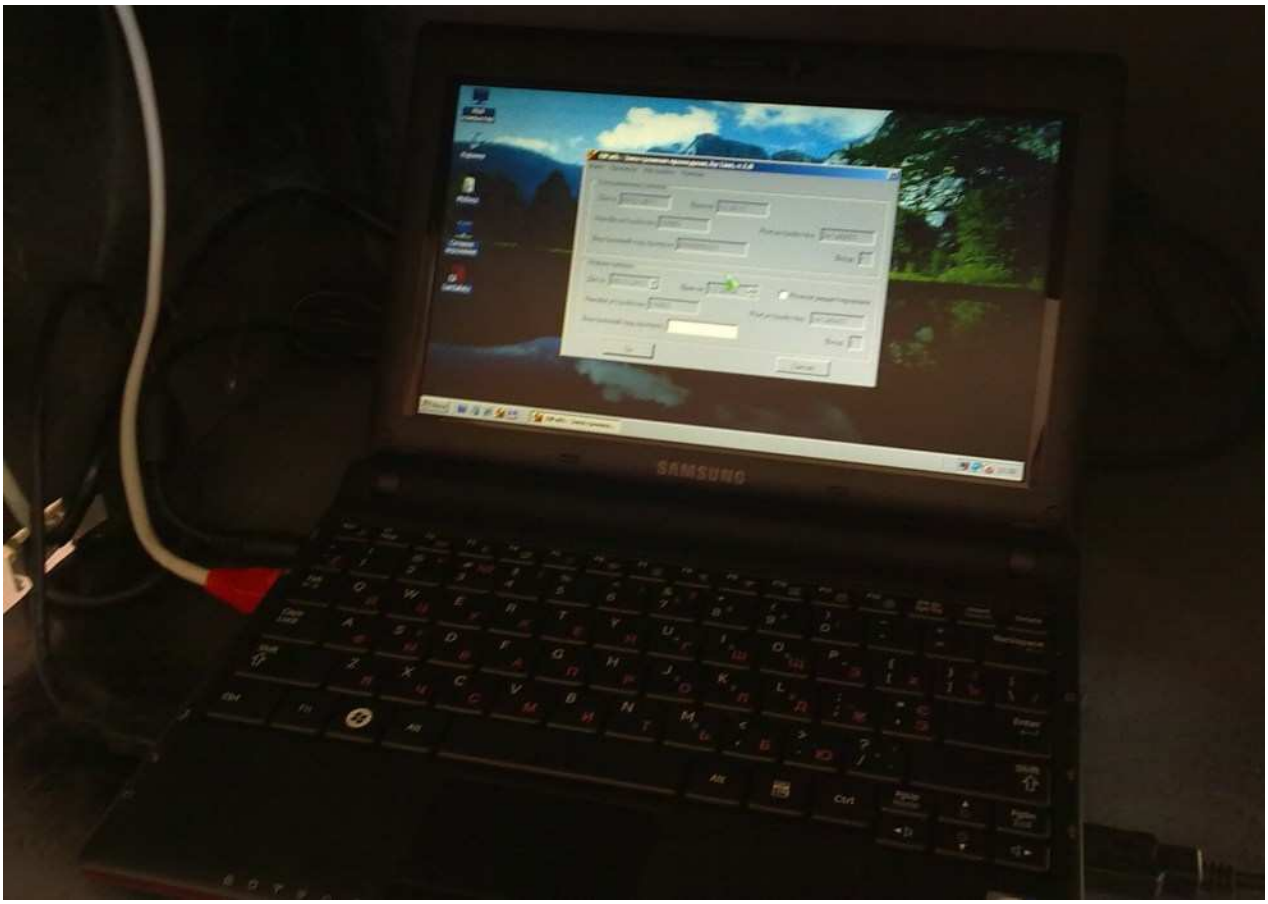

Рис. 6. Нетбук в рабочем режиме (с автоматически загружаемой программой ElPath.exe).

Рис. 7. Общий вид системы учета рабочего времени.

Примечания

1. Помимо обычного способа компиляции вы можете использовать наш командный файл **exe60.cmd**, настроенный под **Visual Studio C++ v. 6.0**. Только настройте его данные под себя, например, системный диск у вас может быть не **E:** как у меня, а **C:** и т.д. Выполнение этого файла генерит командный файл **out.cmd**, запуск которого уже дает необходимый файл **ElPath.exe**. Заметим, что командная генерация дает файл в два раза меньшего размера, чем кодогенерация посредством **MS VC6**. Для примера, представлены оба файла **ElPath.exe** (86016 байт) и **ElPath_cmd.exe** (40960 байт).

2. Первоисточник этой статьи расположен в http://emery-emerald.narod.ru/Others/ElPath.html .