KV Betriebssysteme
Windowsprogrammierung

KV Betriebssysteme
Windowsprogrammierung
Aufbau von Windows-Programmen

Michael Sonntag
Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria
E-Mail: sonntag@fim.uni-linz.ac.at

Grund-Aufbau

Code
- **Der Programmcode, exkl. DLLs**

Daten
- **Statische Initialisierungsdaten für Variablen, ...**

Ressourcen
- **Daten zur Definition der sichtbaren Programmteile:**
  - Dialogfenster, Bitmaps, Strings, …
  - Zum leichteren Austausch, z. B. für Internationalisierung
- **Ressourcen unabhängig vom Programm: Änderung durch Linken**
- **Standard-Ressourcen: Siehe nächste Folie**
- **Custom-Ressource: Beliebig von der Applikation definiert**

Arten von Ressourcen (1)

- **Dialogfenster**
  - **Samt darin enthaltenen Controls (Listboxen, Buttons, …)**
  - **Spezifizierung ist Geräteunabhängig**
- **Menüs**
  - **Beliebig verschachtelte Popup-Menüs**
- **Bitmaps**
  - **Rastergrafiken, die im Programm angezeigt werden können**
- **Icons**
  - **Bitmaps mit einer zusätzlichen Sichtbarkeitsmaske**
- **Accelerators**
  - **Zurodnung von Tastenkürzeln zu Menübefehlen**
- **Stringtabellen**
  - **Mit symbolischen Namen versehene Texte.**

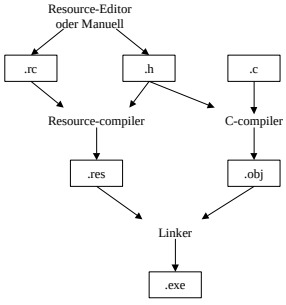
Arten von Ressourcen (2)

- **Cursor**
  - **Bitmaps für Cursors**
- **Font**
  - **Schriftart (Angabe des Dateinamens)**
- **Message-Table**
  - **Spezielle Strings für Ereignis-Logging und Formatierung von Ausgaben**
- **Version**
  - **Informationen zur Version des Programms (Produktversion, Dateiversion, Betriebssystem, …)**

Erzeugen von Ressourcen (1)

- **Ressourcen können in Textdateien (.rc) beschrieben werden, ein Resource-Compiler erzeugt daraus eine binäre Datei (.res), die dann vom Linker zusammen mit dem Code und den Daten zum Exe-File gebunden wird.**
- **Die einzelnen Ressourcen werden über Nummern bzw. Namen angesprochen, die in einer gesonderten Datei gespeichert werden (meist „resource.h“).**
- **LPTSTR MAKEINTRESOURCE(WORD wInteger)**
  Wandelt eine Resource-Identifikationsnummer in einen entsprechenden String um; für Resource-Management-Funktionen hilfreich

Erzeugen von Ressourcen (2)



KV Betriebssysteme
Windowsprogrammierung

Erzeugen von Ressourcen (3)

Händisch
- **NUR für einfache Ressource wie Menüs, Stringtabellen, … geeignet!**

Tools
- **Für alles andere sollte ein Ressourcen-Editor verwendet werden.**
- **Diese erlauben meist auch das Verändern von fertigen Programmen, da sie nicht nur mit Quell-Ressourcen (.rc), sondern auch compilierten (.res) und gelinkten arbeiten können.**

Aufteilung der Informationen

1. Resource.h
- **Definition der Namen der Ressourcen als Konstanten.**
- **Konvention:**
  - IDI_???: Icons
  - IDM_???: Menüs
  - IDB_???: Bitmaps
  - IDD_???: Dialog
  - IDS_???: Strings

2. <Programmname>.rc
- **Eigentliche Definition der Ressourcen, bzw. Spezifikation des Namens der externen Datei**

Einbinden von Resourcen

Verwendung im Programm
- **FormatMessage(...) für Message-Tabellen**
- **LoadAccelerators(...) für Tastenkürzel**
- **LoadImage(…) für Icons, Cursors, Bitmaps und Metafiles**
- **LoadMenu(…) für Menüs**
- **LoadString(…) für Stringtabellen-Einträge**

Beispiel:

```
#include "resource.h"
…
WNDCLASSEX wc;
wc.hIcon = LoadIcon(hInstCurr, MAKEINTRESOURCE(IDI_MAINICON));
```


KV Betriebssysteme Windowsprogrammierung Arten von Ressourcen

Michael Sonntag
Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria
E-Mail: sonntag@fim.uni-linz.ac.at

Icons

nameID ICON dateiname

Definiert ein Icon als den Inhalt einer Datei.

Funktionen:
- **HANDLE LoadImage(„IMAGE_ICON„„): Laden von Icons**
- **BOOL DrawIcon(HDC hDC, int x, int y, HICON hIcon): Icon anzeigen**
- **BOOL GetIconInfo(HICON hIcon, PICONINFO pInfo) : Informationen über das Icon einholen (Hotspot, Zeichen-Bitmap, Masken-Bitmap).**

Menüs - Aufbau (1)

menuID MENU {…}

Eine horizontale Menüzeile mit beliebigen Eiträgen; kann auch Popups enthalten.

Erlaubt:
- **MENUITEM**
- **POPUP**

POPUP: Ein vertikales Menü, das Menüeinträge und weitere Submenüs enthalten kann.
- **POPUP text, optionlist {...}**
  - Optionen: CHECKED, GRAYED, HELP, INACTIVE, MENUBREAK, MENUBARBREAK MENUITEM

KV Betriebssysteme
Windowsprogrammierung

Menüs - Aufbau (2)

MENUITEM
- **MENUITEM SEPARATOR Trennstrich zwischen Einträgen**
- **MENUITEM text, id, optionlist**
  - Optionen: Wie bei POPUP

•Sowohl das Menü als auch alle Menüeinträge (nicht aber die Popups!) benötigen eine Nummer: IDM_????

Menüs - Aufbau (3)

Mit einem "&" kennzeichnet man den folgenden Buchstaben als Tastenkürzel für die Bedienung über die Tastatur. Er wird automatisch unterstrichen angezeigt.
- **Nicht mit Accelerators verwechseln!**
- **Diese werden mit "\t" angefügt, sind aber rein optisch als Hilfe für den Benutzer da. Ihre Funktion ist extra zu definieren und extra zu implementieren. Sie sind auch nicht zwingend an Menüeinträge gebunden.**

Menüs - Events

WM_COMMAND
- **Wenn ein Menüpunkt ausgewählt wurde**
- **Menü-ID ist in LOWORD(wParam) zu finden**

WM_INITMENU:
- **Vor dem Anzeigen zur Aktualisierung durch den Benutzer**

WM_INITMENUPOPUP:
- **Wie oben, aber für Popups**

WM_CONTEXTMENU:
- **Bei Rechts-clicks. Zum öffnen von Kontextmenüs falls gewünscht; sonst an DefWindowProc! Siehe TrackPopupMenuEx(...).**

Menüs - Beispiel (1)

Program.rc:

```
#include "resource.h"
…
IDM_MAINMENU MENU {
        POPUP "&Game" {
                MENUITEM "&new\tCtrl+N", IDM_GAME_NEW
                MENUITEM SEPARATOR
                MENUITEM "E&xit", IDM_GAME_EXIT
        }
        POPUP "&Help" {
                MENUITEM "&About\tF1", IDM_HELP_ABOUT
        }
```


Menüs - Beispiel (2)

Program.c:

```
LRESULT CALLBACK WndProc (HWND hWnd,
    UINT message, WPARAM wParam, LPARAM lParam) {
    switch (message)  {
        case WM_COMMAND:
            switch(LOWORD(wParam)) {
                case IDM_GAME_EXIT: PostQuitMessage(0); break;
                case IDM_HELP_ABOUT: About(); break;
            }
            break;
……
}
int InitApplication(HINSTANCE hInstCurr) {
wc.lpszMenuName=MAKEINTRESOURCE(IDM_MAINMENU);
```


Menüs - API-Funktionen

Ausgewählte Funktionen:
- **GetMenuItemInfo(): Menü-Infos (ID, Typ, State, Submenu, ...)**
- **SetMenuItemInfo(): Menü-Infos festlegen**
- **EnableMenuItem(): Enabled, Disabled, Grayed**
- **GetMenu(): Menü eines Fensters holen**
- **GetSubMenu(): Handle für Submenü holen**
- **LoadMenu(): Menü aus Ressourcen laden**
- **TrackPopupMenuEx(): Popup-Menü darstellen und durchführen**

KV Betriebssysteme
Windowsprogrammierung

Accelerators

- **Accelerators sind Tastenkürzel für Menüeinträge und sonstige Funktionen. Auch wenn sie nicht in Menüs aufscheinen, werden sie identisch behandelt: WM_COMMAND.**
- **Im Programm müssen Accelerators mit LoadAccelerators geladen werden.**
- **Die Hauptschleife ist wie zu verändern; siehe Beispiel. Wenn ein Accelerator-Tastendruck vorliegt (Rückgabe!= 0), darf TranslateMessage und DispatchMessage NICHT mehr aufgerufen werden!**
- **Die Tastenkombination ALT+F4 ist im Systemmenü des Hauptfensters als Kürzel zum Beenden des Programms definiert und kann daher nicht verwendet werden.**

Accelerators - Definition

accTableName ACCELERATORS {
event, idValue, [type, options]
}

event: **“character”, ASCII-Code, virtuelle Bezeichnung**
idValue: **Bezeichner (Code für WM_COMMAND)**
type: **ASCII oder VIRTKEY**
options: **ALT, SHIFT, CONTROL, NOINVERT**

•Bei WM_COMMAND ist HIWORD(wParam)=1, falls das Kommando von einem Accelerator ausgelöst wurde.

Accelerators - Beispiel (1)

Program.rc:

```
IDA_SHORTCUTS ACCELERATORS {
        ”N", IDM_GAME_NEW, VIRTKEY, CONTROL
        VK_F1, IDM_HELP_ABOUT, VIRTKEY
}
```

CTRL+N: **WM_COMMAND+IDM_GAME_NEW**
F1: **WM_COMMAND+IDM_HELP_ABOUT**

Accelerators - Beispiel (2)

Program.c:

```
int WINAPI WinMain (HINSTANCE hInstCurr,
      HINSTANCE hInstPrev, LPSTR lpszCmd, int nCmdShow) {
   HACCEL hAccel;
   hAccel=LoadAccelerators(hInstCurr,
               MAKEINTRESOURCE(IDA_SHORTCUTS));
   while (GetMessage (&msg, NULL, 0, 0)>0) {
      if(!TranslateAccelerator(hWndMain, hAccel, &msg)) {
         TranslateMessage (&msg); DispatchMessage (&msg);
      }
   }
   return msg.wParam;
}
```


Dialoge - Arten

Dialoge sind spezielle Fenster zur Interaktion mit dem Benutzer. Im Gegensatz zu Fenstern sind sie meist nur kurz sichtbar und dienen zum Einstellen von Parametern bzw. für Mitteilungen.

Typen:

- **Modal: Das Programm läuft erst weiter, wenn der Dialog geschlossen wird (Keine Interaktion mit Hauptfenster möglich).**
  - System-Modal: Das gesamte Windows-System wird gesperrt und NUR dieser Dialog ist zugänglich (z. B. Shutdown-Dialog)
- **Nicht-Modal: Es kann beliebig zwischen Hauptfenster und Dialog gewechselt werden. Z. B. Such-Dialoge**
  - Diese sind um einiges komplexer, da meist wechselseitiges Update erforderlich ist!

Dialoge - Elemente

Windows-Elemente:

- CTEXT: Zentrierter statischer Text
- PUSHBUTTON: Normaler Button
- RADIOBUTTON, CHECKBOX: Radio/Check-Button
- GROUPBOX: Rechteck zur Gruppierung (Radio-Buttons!)
- COMBOBOX, LISTBOX: Auswahl aus einer Liste
- EDITTEXT: Texteingabe (Ein-/Mehrzeilig)
- …..

“Custom-Controls”:

- Werden als OCX oder sonstige Dateien (z. B. ActiveX, ...) zur Verfügung gestellt. Viele Hersteller bieten Bibliotheken an.

Tip: Dialog-Editor benutzen!

KV Betriebssysteme
Windowsprogrammierung

Dialoge - Events

WM_COMMAND:

- **Wird von den einzelnen Controls an die Dialog-Fensterprozedur geschickt. LOWORD(wParam) ist die ID des Controls, welches die Nachricht ausgelöst hat (z. B. IDD_ABOUT_OK), HIWORD(wParam) den Subtyp (Control-spezifisch; z. B. BN_CLICKED).**

WM_INITDIALOG:

- **Wird an den Dialog gesendet unmittelbar bevor er angezeigt wird. Typischerweise werden hier die anzuzeigenden Werte initialisiert (Text in Edit-Controls, Radio-/Checkboxes setzen, Listboxen füllen, ...).**
- **TRUE zurückgeben, damit Windows den Fokus automatisch setzt (selbst ein Control aktiviert: FALSE).**

Dialoge - Programmierung (1)

Dialog erzeugen (nur modale!):

- **int DialogBox(HINSTANCE hInst, LPCTSTR lpTemplate, HWND hWndParent, DLGPROC lpDlgFnc)**
  hInst: Instanz des Programmes, zu dem der Dialog gehört
  lpTemplate: Dialog-Ressource (MAKEINTRESOURCE)
  hWndParent: Übergeordnetes Fenster
  lpDlgFnc: Dialog-Fensterprozedur
- **Die Funktion kehrt erst zurück, wenn der Dialog geschlossen wurde (Modaler Dialog!). Rückgabewert: Von EndDialog**

Dialoge - Programmierung (2)

Dialog-Fensterprozedur:

- **Wie jedes Fenster braucht auch ein Dialog eine Fensterprozedur:**
- **BOOL CALLBACK DialogProc(HWND hWndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)**
- **Außer bei WM_INITDIALOG sollte nur dann 0 zurückgegeben werden, wenn die Nachricht nicht behandelt wurde.**
- **DefWindowProc darf NICHT aufgerufen werden! Bei einem Rückgabewert von 0 wird die Nachricht von Windows selbst behandelt.**

Dialoge - Programmierung (3)

Dialog beenden:

- **BOOL EndDialog(HWND hDlg, int nResult)**
  hDlg: Handle des Dialogs
  nResult: Wert, der von DialogBox zurückgegeben werden soll
- **Bei Erfolg wird ein Wert ungleich Null zurückgegeben.**

Dialog-Ressourcen:

- **Es gibt viele Controls; siehe Online-Hilfe oder Dokumentation**
- **Größe und Position werden in logischen Einheiten spezifziert**

Dialoge - Programmierung (4)

Zugriff auf Controls:

- **HWND GetDlgItem(HWND hDlg, int nIDDlgItem)**
  hDlg: Handle des Dialogs
  nIDDlgItem: ID des gewünschten Elements (z. B. IDD_RUN)
- **Über das zurückgegebene Handle können dann Control-spezifische Funktionen aufgerufen werden:**
  - GetWindowText, SetWindowText: Text von Buttons und statischem Text holen bzw. setzen.
  - Status von Check- und Radio-Buttons holen bzw. setzen:
  - Checked=SendMessage(handle,BM_GETCHECK,0,0);
  - SendMessage(handle,BM_SETCHECK,checked,0);

Dialoge - Beispiel (1)

Dialog.rc:

```
IDD_ABOUT DIALOG 10,10,150,70
    CAPTION "About Dialog"
    STYLE DS_MODALFRAME|WS_POPUP|WS_CAPTION|WS_SYSMENU
{
    PUSHBUTTON "&Ok", IDD_ABOUT_OK,30,45,30,14,
            WS_CHILD | WS_VISIBLE | WS_TABSTOP
    PUSHBUTTON "&Cancel", IDD_ABOUT_CANCEL,90,45,30,14,
            WS_CHILD | WS_VISIBLE | WS_TABSTOP
    CTEXT "Version 1.0", -1, 0,24,150,8,
            WS_CHILD | WS_VISIBLE | WS_GROUP
    CTEXT "Ein einfaches Windows-Programm mit Dialog-Box", -1,09,9,150,8,
            WS_CHILD | WS_VISIBLE | WS_GROUP
}
```

KV Betriebssysteme
Windowsprogrammierung

Dialoge - Beispiel (2)

```
BOOL CALLBACK AboutDialogProc(HWND hwndDlg,
        UINT uMsg, WPARAM wParam, LPARAM lParam) {
    switch(uMsg) {
        case WM_INITDIALOG: return TRUE;
        case WM_COMMAND:
            if(wParam==IDD_ABOUT_OK) {
                EndDialog(hwndDlg, 0); return TRUE; }
            if(wParam==IDD_ABOUT_CANCEL) {
                EndDialog(hwndDlg, 1); return TRUE; }
        break;
        case WM_CLOSE: EndDialog(hwndDlg, 1); return TRUE;
    }
    return FALSE;
}
```


Dialoge - Beispiel (3)

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT message,
            WPARAM wParam, LPARAM lParam) {
    switch (message) {
        case WM_COMMAND:
            switch(LOWORD(wParam)) {
                case IDM_HELP_ABOUT:
                    res=DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUT),
                                hWndMain,AboutDialogProc);
                    sprintf(str,"Der Dialog gab den Wert \"%d\" zurück!",res);
                    MessageBox(hWnd,str,"Rückgabewert des Dialogs",
                                MB_OK | MB_ICONINFORMATION);
                    return res;
            }
```


Dialoge - Beispiel (4)

Ergebnis:

Bitmaps

Bitmaps sollten als externe Datei gespeichert werden und nur in die RC-Datei includiert werden:

nameID BITMAP filename

Verwendung im Programm:

- **HANDLE LoadImage(HINSTANCE hInst, LPCTSTR lpszName, IMAGE_BITMAP, int cx, int cy, UINT fuLoad)**
  - hInst: Programminstanz
  - lpszName: Name der Bitmap, Bitmap-Identifier oder Dateiname
  - cx, cy: Brerite/Höhe der Bitmap (auf 0 setzen für phys. Größe)
  - fuLoad: Lade-Parameter (LR_DEFAULTSIZE, LR_LOADFROMFILE, ...)

Bitmaps - Beispiel (1)

```
case WM_PAINT: {
        hDC=BeginPaint(hWndMain, &ps);
        GetClientRect(hWndMain, &r);
        memDC=CreateCompatibleDC(hDC);
        oldBmp=SelectObject(memDC, bkground);
        BitBlt(hDC, 0, 0, r.right, r.bottom, memDC, 0, 0, SRCCOPY);
        SelectObject(memDC, oldBmp);
        DeleteDC(memDC);
        EndPaint(hWndMain, &ps);
}
int InitInstance(HINSTANCE hInst, int nCmdShow) {
        BITMAP bmp;
        bkground=LoadImage(hInstCurr, MAKEINTRESOURCE(IDB_BGR),
                    IMAGE_BITMAP, 0, 0, LR_DEFAULTCOLOR);
        if(!bkground) return FALSE;
```


Bitmaps - Beispiel (2)

```
if(!GetObject(bkground, sizeof(bmp), &bmp)) {
        DeleteObject(bkground); return FALSE; }
hWndMain = CreateWindow(szWndClassName,
        "Fenster mit Bitmap-Hintergrund",
        /* Nicht größenveränderbar! */
        WS_DLGFRAME | WS_MINIMIZEBOX | WS_SYSMENU,
        CW_USEDEFAULT, CW_USEDEFAULT,
        /* Horizontal: Zwei mal Rahmenbreite addieren */
        bmp.bmWidth+2*GetSystemMetrics(SM_CXFIXEDFRAME),
        /* Vertikal: Zwei mal Rahmenbreite und Titelzeilenhöhe addieren */
        bmp.bmHeight+2*GetSystemMetrics(SM_CYFIXEDFRAME)+
                    GetSystemMetrics(SM_CYCAPTION),
        0, 0, hInstCurr, NULL);
```

KV Betriebssysteme
Windowsprogrammierung
Timer

Michael Sonntag
Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria
E-Mail: sonntag@fim.uni-linz.ac.at

Timer

Mit Timern können periodisch Benachrichtigungen ausgelöst werden:

- **WM_TIMER**
- **Callback-Prozedur**

Die Funktion wird regelmäßig ausgelöst, bis der Timer gelöscht wird.

Timer - API-Funktionen

UINT SetTimer(HWND hWnd, UINT nIdEvent,
UINT uElapse,
TIMERPROC lpFunc)

- **uElapse: Timeout in Millisekunden**
- **lpTimerFunc angegeben: Prozedur wird aufgerufen**
- **lpTimerFunc==NULL: an das Fenster hWnd wird die Nachricht WM_TIMER mit dem Parameter nIdEvent (als wParam) geschickt**
- **Rückgabewert: ID des Timers (für KillTimer!)**

Timer - API-Funktionen

BOOL KillTimer(HWND hWnd, UINT uIdEvent)

- **hWnd: Muß gleich dem Aufruf von SetTimer sein.**
- **uIdEvent: Wenn hWnd!=NULL, muß dies der Parameter nIdEvent von SetTimer sein, ansonsten der Rückgabewert dieser Prozedur.**
- **Der Rückgabewert ist ungleich 0 bei Erfolg.**

Timer - Beispiel

Jede Sekunde ein Pieps-Ton:

```
#define ID_TIMER      123
….
In WndProc:
   case WM_TIMER: MessageBeep(MB_OK); break;
in WinMain:
   SetTimer(hWndMain, ID_TIMER, 1000, NULL);
   while(GetMessage(&msg, NULL, 0, 0)>0) {}
   KillTimer(hWndMain, ID_TIMER);
```


KV Betriebssysteme
Windowsprogrammierung
Dateien

Michael Sonntag
Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria
E-Mail: sonntag@fim.uni-linz.ac.at

KV Betriebssysteme
Windowsprogrammierung

Dateioperationen

Datei:
- **Erzeugen**
- **Öffnen**
- **Lesen**
- **Schreiben**
- **Schließen**

Temporäre Dateien

Kopieren/Verschieben/Löschen von Dateien

Locking/Unlocking

Verzeichnisse

Dateien öffnen/erzeugen

HANDLE CreateFile(LPCTSTR lpFileName,
DWORD dwDesiredAccess, DWORD shareMode,
LPSECURITY_ATTRIBUTES lpSecAttrib,
DWORD dwCreationDisposition,
DWORD dwFlagsAndAttributes,
HANDLE hTemplateFile);

Wird verwendet für: Erzeugen und Öffnen

Für Objekte vom Typ: Datei, Pipe, Mailslot, Netzwerkressourcen, Laufwerken (nur NT), Konsolen, Verzeichnissen (nur öffnen)

Parameter von CreateFile (1)

- **lpFileName: Null-terminierter String des Dateinamens**
- **dwDesiredAccess: Wie zugegriffen werden soll**
  - 0 (nur Laufwerks-Attribute lesen), GENERIC_READ, GENERIC_WRITE
- **shareMode: 0 (Kein anderer Zugriff)**
  - FILE_SHARE_READ: Lesezugriff erlaubt
  - FILE_SHARE_WRITE: Schreibzugriff erlaubt
  - FILE_SHARE_DELETE: Löschzugriff erlaubt
- **lpSecAttrib: Ob Child-Prozesse das Handle verwenden können (NULL: nur dieser Prozess)**

Parameter von CreateFile (2)

- **dwCreationDisposition: Modus für existierende/nicht existierende Dateien**
  - CREATE_NEW, CREATE_ALWAYS, OPEN_EXISTING, OPEN_ALWAYS, TRUNCATE_EXISTING
- **dwFlagsAndAttributes: Archiv, Versteckt, System, …**
  - Zusätzlich: Buffering, Delete on close, Asynchroner Zugriff, ...
- **hTemplateFile: Handle auf eine Template-Datei, von der Attribute (normale und erweiterte) übernommen werden**
- **Rückgabewert: Handle auf die Datei (kann auch 0 sein!)**
  - **Fehler: INVALID_HANDLE_VALUE**

Dateien lesen (1)

BOOL ReadFile(HANDLE hFile, LPVOID lpBuffer,
DWORD nBytesToRead, LPDWORD lpBytesRead,
LPOVERLAPPED lpOverlapped)

hFile: Handle auf die Datei

lpBuffer: Zeiger auf den Buffer, in den gelesen wird

nBytesToRead: Wieviele Bytes gelesen werden sollen

lpBytesRead: Anzahl der gelesenen Bytes

lpOverlapped: Für asynchrones lesen

Rückgabewert: Ungleich 0 bei Erfolg

Datei lesen (2)

Lesen beginnt am Dateizeiger, der um die tatsächlich gelesenen Bytes aktualisiert wird.
- **Asynchrones Lesen: KEINE Aktualisierung des Dateizeigers!**

Der Buffer darf während des Lesens nicht verändert werden (insbesondere bei asynchronem Zugriff!)

Dateiende:
- **Synchroner Zugriff: Rückgabewert TRUE, lpBytesRead ist 0**

KV Betriebssysteme
Windowsprogrammierung

Dateien schreiben (1)

BOOL WriteFile(HANDLE hFile, LPCVOID lpBuffer,
DWORD nBytesToWrite,
LPDWORD lpBytesWritten,
LPOVERLAPPED lpOverlapped)

hFile: Handle auf die Datei

lpBuffer: Zeiger auf Buffer, aus dem geschrieben wird

nBytesToWrite: Anzahl der zu schreibenden Bytes

lpBytesWritten: Zeiger auf tatsächlich geschriebene Bytes

lpOverlapped: Für asynchrones schreiben

Rückgabewert: Ungleich 0 bei Erfolg

Dateien schreiben (2)

Schreiben beginnt am Dateizeiger, der um die tatsächlich geschriebenen Bytes aktualisiert wird.

- **Asynchrones Schreiben: KEINE Aktualisierung des Dateizeigers!**

Der Buffer darf während des Schreibens nicht verändert werden (insbesondere bei asynchronem Zugriff!)

Null Bytes zu schreiben wird einfach ignoriert (NOP); dies bedeutet KEIN Truncate oder EOF

Zum Beenden der Datei:
BOOL SetEndOfFile(HANDLE hFile)

Dateizeiger setzen

DWORD SetFilePointer(HANDLE hFile, LONG lDist,
PLONG lpDistHigh, DWORD dwMoveMethod)

hFile: Datei-Handle

lDist: Abstand: + Þ nach hinten, - Þ nach vorn

lpDistHigh: NULL oder höherwertige 32 Bit für Abstand

dwMoveMethod: Anfangspunkt für Abstand

- **FILE_BEGIN, FILE_CURRENT oder FILE_END**

Rückgabewert: Neuer Dateizeiger

- **Bei Fehlern: Kompliziert (0xffffffff bei lpDistHigh==NULL)!**

Dateien schließen

BOOL CloseHandle(HANDLE hObject)

•hObject: Handle auf ein offenes Objekt

•Rückgabewert: Ungleich 0 bei Erfolg

•Handles dürfen nur einmal geschlossen werden!

Temporäre Dateien

UINT GetTempFileName(LPCTSTR lpPathName,
LPCTSTR lpPrefix, UINT uUnique,
LPTSTR lpTempFileName)

lpPathName: Pfad für die temporäre Datei
(Meist: "." oder GetTempPath(...))

lpPrefix: Ersten drei Zeichen des Namens

uUnique: 0…Eindeutiger String wird erzeugt
!=0…Diese Nummer wird verwendet

lpTempFileName: Buffer für den Namen der Datei

- Buffer-Länge: MAX_PATH

Temporäre Dateien

Wird keine Nummer angegeben (uUnique=0), so wird die Datei erzeugt (und wieder geschlossen!).

Rückgabewert ist uUnique bzw. die verwendete Zahl

Die Extension ist immer ".TMP"

Temporäre Dateien werden NICHT automatisch gelöscht.

Dateiname:

- **path\preuuuu.TMP**
  - path Pfad aus lpPathName
  - pre Die ersten drei Zeichen von lpPrefix
  - uuuu Hexadezimalwert von uUnique (bzw. aus Systemzeit erstellt)

KV Betriebssysteme
Windowsprogrammierung

Kopieren

BOOL CopyFile(LPCTSRT source, LPCTSTR dest, BOOL bFailIfExist)

source, dest: Nullterminierte Dateinamen

bFailIfExist: Fehler wenn das Ziel schon existiert

Rückgabewert: Ungleich Null bei Erfolg

Die Datei muß geschlossen oder für Nur-Lese-Zugriff geöffnet sein!

Sicherheitsattribute werden nicht kopiert, normale Datei-Attribute schon.

Siehe auch: CopyFileEx(…) !

Verschieben

BOOL MoveFile(LPCTSRT source, LPCTSTR dest)

source, dest: Nullterminierte Dateinamen

Rückgabewert: Ungleich Null bei Erfolg

Die Datei muß geschlossen sein!

Siehe auch: MoveFileEx(…) !

Auch für Verzechnisse, aber NUR auf dem selben Laufwerk.

Löschen

BOOL DeleteFile(LPCTSTR lpFilename)

lpFilename: Nullterminierter Dateiname

Rückgabewert: Ungleich Null bei Erfolg

Die Datei muß geschlossen sein!

Locking

Eine Datei kann von mehreren Programmen gleichzeitig geschrieben werden. Um Probleme zu verhindern kann man (=USER!) Locking verwenden.

Gesperrte Bereiche können von anderen Prozessen weder gelesen noch geschrieben werden.

Man verwendet dazu:

LockFile, LockFileEx

UnlockFile, UnlockFileEx

Gesperrte Bereiche müssen 1:1 freigegeben werden.

Verzeichnisse

CreateDirectory: Verzeichnis erzeugen

RemoveDirectory: Verzeichnis löschen (muß leer sein!)

GetCurrentDirectory: Verzeichnis abfragen

SetCurrentDirectory: Verzeichnis wechseln

Diese Funktionen arbeiten alle auf Dateinamen, nicht auf Handles!

Verzeichnis-Inhalt

HANDLE FindFirstFile(LPCTSTR lpFileName, LPWIN32_FIND_DATA lpFindData)

- **Sucht nur nach Wildcards; lpFindData ist das erste Ergebnis.**

BOOL FindNextFile(HANDLE hFindFile, LPWIN32_FIND_DATA lpFindData)

- **Das nächste Ergebnis (Rückgabe 0 wenn keine weiteren Dateien)**

BOOL FindClose(HANDLE hFindFile)

- **Beenden der Suche**

FindFirstFileEx erlaubt komplexere Suchen!