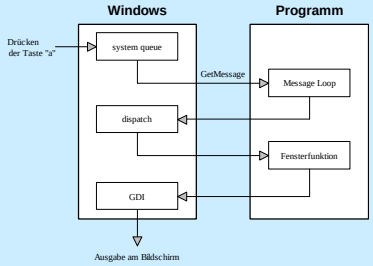
KV Betriebssysteme
Windowsprogrammierung

KV Betriebssysteme
Windowsprogrammierung

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

Windows - Programmiermodell



Aufbau eines Windowsprogramm

Ein einfaches Windowsprogramm, das ein Fenster öffnet, besteht aus zwei Funktionen:

- **WinMain ("Hauptprogramm")**
  - Globale Initialisierung
  - Lokale Initialisierung
  - "message loop"
- **Fensterfunktion**
  - Darstellung des Fensters

WinMain

```
int WINAPI WinMain (  HINSTANCE hInstCurr,
                      HINSTANCE hInstPrev,
                      LPSTR     lpszCmd,
                      int       nCmdShow) {
 /* -- Globale Initialisierung -- */
 /* -- Lokale Initialisierung -- */
 /* -- „Message Loop“ -- */
 return 0;
}
```

```
typedef void *HINSTANCE;
typedef char *LPSTR;
```


Initialisierung

Instanzen

- **Ein Programm kann mehrmals aufgerufen werden.**
- **Dabei wird nur einmal der Code geladen.**
- **Die Daten werden für jede Instanz geladen.**

Globale Initialisierung

- **Initialisierung unabhängig von einer bestimmten Instanz**
- **Sie wird in der Regel nur einmal beim Aufruf der ersten Instanz durchgeführt.**

Lokale Initialisierung

- **Initialisierung einer bestimmten Instanz**

Initialisierung (2)

```
int WINAPI WinMain
  (hInstCurr, hInstPrev, lpszCmd, nCmdShow) {
 /* -- Globale Initialisierung -- */
 if (!hInstPrev)                          /* erste Instanz? */
   if (!InitApplication(hInstCurr)) return 0;
 /* -- Lokale Initialisierung -- */
 if (!InitInstance(hInstCurr, nCmdShow)) return 0;
 /* -- „Message Loop“ -- */
 return 0;
}
```

KV Betriebssysteme
Windowsprogrammierung

Initialisierung (3)

Erkennung der ersten Instanz
- Die erste Instanz einer Applikation erkennt man am Parameter hInstPrev. Er ist NULL, wenn das Programm das erste Mal aufgerufen wird.

Globale Initialisierung
- Funktion: int InitApplication(HINSTANCE hInstCurr);
- Registrierung der benötigten Fensterklassen

Lokale Initialisierung (Funktion InitInstance)
- int InitInstance(HINSTANCE hInstCurr, int nCmdShow);
- Öffnen des Hauptfensters der Applikation

Fensterklassen

Was ist eine Fensterklasse (Window Class)?
- Menge von Eigenschaften
- Template für neue Fenster
- Fensterklassen sind Prozeß-spezifisch.
- Alle Fenster einer Klasse teilen sich eine Fensterfunktion.

Was ist eine Fensterfunktion (Window Procedure)?
- Verarbeitet Nachrichten für alle Fenster seiner Klasse.
- Kontrolliert das Verhalten und Erscheinungsbild aller Fenster seiner Klasse. Daher haben alle Fenster einer Klasse dasselbe Erscheinungsbild und Verhalten.

Arbeiten mit Fensterklassen

1. Registrierung einer Fensterklasse
- Funktion:
  ATOM RegisterClass(CONST WNDCLASS *lpWndClass);
- Durch die Registrierung bekommt eine Fensterklasse einen Namen, eine Fensterfunktion und Stile, wie z.B. Mauscursor, Hintergrundfarbe, ...

2. Öffnen eines Fensters
- Funktion:
  HWND CreateWindow(LPCTSTR lpClassName, ...);
- Mit der Funktion CreateWindow wird ein Fenster einer bestimmten Klasse angelegt.

Struktur WNDCLASS

```
typedef struct _WNDCLASS {
  UINT      style;          /* Klassenstile              */
  WNDPROC   lpfnWndProc;    /* Fensterfunktion           */
  int       cbClsExtra;     /* Extra Klassenspeicher     */
  int       cbWndExtra;     /* Extra Fensterspeicher     */
  HINSTANCE hInstance;      /* Instanz                   */
  HICON     hIcon;          /* Icon, NULL -> kein I.     */
  HCURSOR   hCursor;        /* Mauscursor, NULL          */
  HBRUSH    hbrBackground;  /* Hintergrundfarbe          */
  LPCTSTR   lpszMenuName;   /* Menüname, NULL            */
  LPCTSTR   lpszClassName;  /* Klassenname               */
} WNDCLASS, *PWNDCLASS;
```


Globale Initialisierung - Beispiel

```
static char szWndClassName[] = "CHelloWorld";

static int InitApplication(HINSTANCE hInstCurr) {
    WNDCLASS wc;   /* window class */
    wc.style              = CS_HREDRAW | CS_VREDRAW ;
    wc.lpfnWndProc        = (WNDPROC)WndProc ;
    wc.cbClsExtra         = 0 ;
    wc.cbWndExtra         = 0 ;
    wc.hInstance          = hInstCurr ;
    wc.hIcon              = LoadIcon(NULL, IDI_WINLOGO);
    wc.hCursor            = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground      = (HBRUSH)(COLOR_WINDOW + 1);
    wc.lpszMenuName       = NULL;
    wc.lpszClassName      = szWndClassName;
    return RegisterClass(&wc);
}
```


Fensterklassen

Zusammenfassung
- Eine Fensterklasse faßt die gemeinsamen Eigenschaften aller zu dieser Klasse gehörenden Fenster zusammen. Dazu gehört zum Beispiel auch die Ereignisverarbeitungsprozedur.
- Das bedeutet, daß alle Fenster einer Klasse im wesentlichen dieselbe Funktionalität und dasselbe Aussehen haben.
- Windows stellt einige Standardklassen (Button, Listbox, Combobox, Eingabefeld, Radio Button, ...) zur Verfügung. Darüber hinaus kann jedes Programm eigene Fensterklassen anlegen.

KV Betriebssysteme
Windowsprogrammierung

CreateWindow

```
HWND CreateWindow(
  LPCTSTR     lpClassName,     // registered class name
  LPCTSTR     lpWindowName, // window name
  DWORD       dwStyle,         // window style
  int         x,               // horizontal position of window
  int         y,               // vertical position of window
  int         nWidth,          // window width
  int         nHeight,         // window height
  HWND        hWndParent,// handle to parent or owner window
  HMENU       hMenu,           // menu handle or child identifier
  HINSTANCE   hInstance,       // handle to application instance
  LPVOID      lpParam          // window-creation data
);
```


Lokale Initialisierung

```
static int InitInstance(HINSTANCE hInstCurr, int nCmdShow) {

   HWND hWnd;                                  // Fenster-Handle, eindeutige ID
   hWnd = CreateWindow(szWndClassName,         // Klassenname
       "Hello World",                          // Fenstertitel
       WS_OVERLAPPEDWINDOW,                    // Fensterstil
       CW_USEDEFAULT, CW_USEDEFAULT,           // x,y-Koordinate
       CW_USEDEFAULT, CW_USEDEFAULT,           // Breite und Höhe
       NULL, NULL, hInstCurr, NULL);           // Parent, Menü, Instanz, Zusatz
   if (hWnd) {
       ShowWindow(hWnd, nCmdShow);             // Fenster anzeigen
       UpdateWindow(hWnd);                     // Fensterinhalt erneuern
       return TRUE;
   }
   return FALSE;
}
```


„Message Loop“

```
int WINAPI WinMain ( hInstCurr, hInstPrev,
                     lpszCmd, nCmdShow) {
   MSG msg ;

   ... (Globale und lokale Initialisierung) ...

   while (GetMessage (&msg, NULL, 0, 0)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }
   return msg.wParam;
}
```


„Message Loop“

Bestandteile

- **GetMessage liest die nächste Nachricht aus der Warteschlange.**
- **TranslateMessage übersetzt "virtual key messages" in "character messages" (z.B. Kombinationen von WM_KEYDOWN und WM_KEYUP).**
- **DispatchMessage sendet die Nachricht an das zuständige Fenster (=> Fensterfunktion).**
- **Die Schleife wird beendet, wenn GetMessage 0 zurückliefert. Das ist der Fall, wenn die Nachricht "WM_QUIT" in der Warteschlange steht.**

Fensterfunktion

Schnittstelle

- **Jedes Fenster (eig. Fensterklasse) hat eine Fensterfunktion.**
- **Der Funktionsname kann frei gewählt werden.**
- **LRESULT WINAPI WndProc(**

```
      HWND hwnd,          /* handle of window          */
      UINT uMsg,          /* message identifier        */
      WPARAM wParam,      /* 1st message parameter     */
      LPARAM lParam);     /* 2nd message parameter     */
```

(The WINAPI modifier is used because the window procedure must be declared with the standard call calling convention.)

Fensterfunktion (2)

Aufgabe

- **Empfang von Nachrichten**
- **„Geeignete“ Reaktion auf die Nachrichten**
  - Man kann auf die Nachricht in seiner Fensterfunktion reagieren oder
  - sie an die „**Default Window Procedure**“ (Standardfunktion) weiterleiten. In den Standardfunktionen ist zum Beispiel das Verschieben eines Fensters oder das Verkleinern und Vergrößern implementiert.

KV Betriebssysteme
Windowsprogrammierung

Fensterfunktion (3)

```
LRESULT WINAPI WndProc (HWND ...) {

    switch (message) {

        case WM_DESTROY: {
            PostQuitMessage (0) ; /* post WM_QUIT */
            return 0;
        }

        default:
            return DefWindowProc (hwnd, message, wParam, lParam) ;
    }
}
```


Standardfunktionen

Funktion	ist vorgesehen für
DefDlgProc	Dialoge
DefFrameProc	MDI Frame Window
DefMDIChildProc	MDI Client Window
DefScreenSaverProc	Screen Saver
DefWindowProc	"normales" Fenster

Nachrichten

WM_DESTROY:
- Das Fenster wird "zerstört". Es ist eine der letzten Nachrichten, bevor das Fenster aufhört zu existieren.

WM_MOUSEMOVE:
- Die Maus wurde über das Fenster bewegt.

WM_MOVE:
- Die Position des Fensters hat sich geändert.

WM_PAINT:
- Inhalt des Fensters soll neu gezeichnet werden.

WM_QUIT:
- Das Programm soll beendet werden.

WM_SIZE:
- Die Größe des Fensters hat sich geändert.

Nachrichten (2)

WM_DESTROY
- keine Parameter

WM_MOUSEMOVE
- wParam = key flags
- LOWORD(lParam) = hor. Pos.
- HIWORD(lParam) = ver. Pos.

WM_MOVE
- LOWORD(lParam) = hor. Pos.
- HIWORD(lParam) = vert. Pos.

WM_PAINT
- keine

WM_QUIT
- wParam = exit code

WM_SIZE
- wParam = sizing-type flag
- LOWORD(lParam) = Breite
- HIWORD(lParam) = Höhe

Beispiel

Das Beispiel öffnet ein Fenster, in dem der Text "Hello World" angezeigt wird.

Dazu wird im Programm eine neue Fensterklasse "CHelloWorld" angelegt, dessen Fensterfunktion
- beim Schließen des Fensters das Programm beendet und
- bei WM_PAINT den Text "Hello World" ausgibt.

WHello.c