Vorlesungsmodul Systemprogrammierung II
- VorlMod SysProg2 -

Matthias Ansorg

10. Oktober 2003 bis 28. März 2005

Zusammenfassung

Studentische Mitschrift zur Vorlesung Systemprogrammierung 2 bei Prof. Dr. Bachmann (Sommersemester 2004) im Studiengang Informatik an der Fachhochschule Gießen-Friedberg.

- **Bezugsquelle:** Die vorliegende studentische Mitschrift steht im Internet zum Download bereit. Quelle: Persönliche Homepage Matthias Ansorg :: InformatikDiplom http://matthias.ansorgs.de/InformatikAusbldg/.
- **Lizenz:** Diese studentische Mitschrift ist public domain, darf also ohne Einschränkungen oder Quellenangabe für jeden beliebigen Zweck benutzt werden, kommerziell und nichtkommerziell; jedoch enthält sie keinerlei Garantien für Richtigkeit oder Eignung oder sonst irgendetwas, weder explizit noch implizit. Das Risiko der Nutzung dieser studentischen Mitschrift liegt allein beim Nutzer selbst. Einschränkend sind außerdem die Urheberrechte der angegebenen Quellen zu beachten.
- **Korrekturen und Feedback:** Fehler zur Verbesserung in zukünftigen Versionen, sonstige Verbesserungsvorschläge und Wünsche bitte dem Autor per e-mail mitteilen: Matthias Ansorg <mailto:matthias@ansorgs.de>.
- **Format:** Die vorliegende studentische Mitschrift wurde mit dem Programm LyX (graphisches Frontend zu LaTeX) unter Linux geschrieben und mit pdfLaTeXals pdf-Datei erstellt. Grafiken wurden mit dem Programm xfig unter Linux erstellt und als pdf-Dateien exportiert.
- **Dozent:** Prof. Dr. Bachmann.
- **Verwendete Quellen:**
- **Klausur:**
  - Die Klausur hat keine formalen Teilnahmevoraussetzungen wie etwa Hausübungen.
  - Wer die Praktikumsaufgaben macht, kann dadurch max. 10% Bonuspunkte zur Klausur erwerben.
  - Die Klausur wird vor den Semesterferien geschrieben.
  - Wer an den Übungen teilnimmt und das Skript liest, sollte die Klausur bestehen können.
  - 3 Wochen vor Semesterende wird entschieden, ob die Klausur mit oder ohne Hilfsmittel geschrieben wird. Bei einer Klausur ohne Hilfsmittel werden die zu verwendenden Funktionen in der Aufgabenstellung zur Verfügung gestellt und müssen dann nur noch »in die richtige Reihenfolge« gebracht werden. Bei einer Klausur mit Hilfsmitteln muss man diese Funktionen selbst in der mitgebrachten Dokumentation suchen.

– Alte Klausuren stehen im Internet und bei der Fachschaft zur Verfügung. Gegen Semesterende wird eine zusätzliche alte Klausur zur Verfügung gestellt.
– Die Klausur wird vom Stil her gleich den Klausuren der vergangenen Semester sein. Stets müssen Fragen zu ereignisgesteuerter Programmierung beantwortet werden.

Inhaltsverzeichnis

1 Organisation

- Stoff dieser Veranstaltung bei Prof. Bachmann ist hauptsächlich die ereignisgesteuerte Programmierung unter Windows. Dabei werden die Übersetzungs-Werkzeuge verwendet, mit denen auch Windows selbst entwickelt wurde: Compiler und Linker aus der VisualStudio- und C#-Umgebung. Diese sind auf allen Windows-Rechnern der FH Gießen installiert. Wird man im Förderverein Mitglied, erhält man billig eine Lizenz (Key) für diese Werkzeuge und kann sich die passenden ISO-Images vom Server exterminator herunterladen.
- In Vorlesung und Praktika besteht keine Anwesenheitspflicht.
- Prof. Bachmann stellt die Praktikumsaufgaben auf seiner Homepage zur Verfügung. Die fertige Lösung einer Praktikumsaufgabe muss man spätestens an dem Termin vorführen, zu dem die nächste Praktikumsaufgabe ausgeteilt wird. In einer Liste werden dann 0, 1 oder 2 Punkte pro Praktikumsaufgabe eingetragen. Im Semester wird es etwa 8 Praktikumsaufgaben geben.
- Es gibt ein Anmeldeverfahren für die Übungen. Es hat einzig den Zweck, dass Prof. Bachmann die Teilnehmerdaten erfährt.
- In den Übungen dürfen eigene Laptops verwendet werden.

2 Einführung

Was ist Systemprogrammierung? Es ist ein strittiger Begriff. Für diese Veranstaltung bedeutet er: wir beschäftigen uns mit Software-Plattformen und -Umgebungen:

- Aspekte ihrer Struktur und Funktionsweise.
- Softwareentwicklung unter ihnen.
- Exemplarische Betrachtung von Windows.

Begriffe:

Software-Plattform Die »tiefste«, d.i. hardwarenächste genutzte Software-Schicht. Das ist i.d.R. das Betriebssystem, außer bei embedded Systems.

Software-Umgebung Satz von Programmen, die für eine bestimmte Nutzung die Kommunikation mit der Plattform übernehmen. Beispiele: OpenGL, frühe Windows-Versionen. Die frühen Windows-Versionen waren keine Plattform, weil sie auf DOS als unterster Schicht basierten.

Systemprogrammierung ist allgemein die Erstellug von Programmen, die den Ablauf »systeminterner« Vorgänge regeln.

Programm ist die Planung einer Reige von Handlungen (inkl. Teilhandlungen und Details), die auf ein einheitliches Ziel hinsteuern.

System Als System bezeichnen wir

- Eine Menge von Komponenten (Gegenstände, Individuen, Ideen)
- die untereinander in einer kausalen (beinhaltet auch statistische) Wechselwirkung stehen
- und von ihrer Umwelt entweder als abgeschlossen oder als in einer wohldefinierten Beziehung stehend betrachtet werden können
- sowie die Gesamtheit der unter ihnen herrschenden Beziehungen.

3 Dinge die man auswendig wissen sollte

- `BOOL EndDialog(HWND hDlg, INT_PTR nResult);`
  Zerstört einen modalen Dialog, d.i. einen der mit der `DialogBox()`-Funktion oder einer ihrer Varianten erzeugt wurde.
- `BOOL DestroyWindow( HWND hWnd );`
  Zerstört Fenster und nichtmodale Dialoge, das sind Dialoge die mit der `CreateDialog()`-Funktion erzeugt wurden.
- `INT_PTR` ist ein Integer
- `BeginPaint(HWND hWnd, PAINTSTRUCT * ps);`
- `EndPaint(HWND hWnd, PAINTSTRUCT * ps);`
- `GetWindowRect(HWND hwnd, LPRECT lprect);`

- `typedef UINT_PTR WPARAM; //word parameter`
  Nur aus historischen Gründen heißt dieser Parameter »word parameter« - er war bei Windows 3.11 tatsächlich 16 bit groß, heute aber 32 bit. Beispiel: bei `WM_COMMAND` enthält `LOWORD(wParam)` den Identfizierer des auslösenden Fensters.

- `typedef unsigned int UINT_PTR;`
  `_PTR` bedeutet »for pointer arithmetic«. Solche Integer-Typen werden benutzt, um Offsets, die sich z.B. bei der Subtraktion und Addition von Pointern ergeben, abzuspeichern. Dazu werden diese Offsets vom Pointer-Typ in solch einen Int-Typ gecastet. Auch Handles sind Int-Zahlen, die als Offset zu Pointern beim Verweis in eine Tabelle dienen. Logisch sind `_PTR`-Integer und Handles also verwandt, wenn auch ein Handle als void-Pointer definiert ist.

- `typedef LONG_PTR LPARAM; //long parameter`

- `HWND GetDlgItem( HWND hDlg, int nIDDlgItem );`

- Alle Makros für Controls wie etwa `Edit_SetText(hctl, lpsz)` bekommen als erstes Element das Handle des Controls, nicht die Ressourcen-ID.

- `LRESULT SendMessage( HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam );`

- Modal und nichtmodal. Jede Dialogbox hat eine eigene Nachrichtenbehandlungsroutine. Was also ist der Unterschied zwischen modalen und nichtmodalen Dialogboxen? Bei nichtmodalen Dialogboxen empfängt die Nachrichtenbehandlungsroutine der Applikation *alle* Nachrichten und ist dafür verantwortlich, Nachrichten an nichtmodale Dialoge weiterzuleiten, die für diese bestimmt sind. Das geschieht in `WindowProc()` geschieht:

  ```
  while (GetMessage(&msg, NULL, 0, 0)) {
    if (!IsWindow(hwndGoto) || !IsDialogMessage(hwndGoto, &msg)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
  }
  ```

  Bei modalen Dialogen empfängt die Nachrichtenbehandlungsroutine des Dialogs selbst *alle* Nachrichten, bis der Benutzer die Dialogbox schließt. Diese Nachrichtenbehandlungsroutine ist dabei `DialogBox()`, die Funktion mit der der Dialog auch erzeugt wird. Keine Nachricht erreicht mehr das Hauptfenster, solange der Dialog angezeigt wird. Auch kann die `DialogBox()`-Funktion nicht geändert werden, sie wird von Windows bereitgestellt. Wegen dieser speziellen, systemdefinierten Nachrichtenschleife ist bei modalen Dialogen auch ein spezielles Beenden mit `EndDialog()` statt direkt per `DestroyWindow()` nötig.
  Nichtmodale Dialogboxen werden mit `CreateDialog()` erzeugt, der Name ist parallel zu `CreateWindow()`, womit ja nichtmodale Fenster erzeugt werden, die ihre Nachrichten mit `DispatchMessage()` aus der Haupt-Nachrichtenschleife der Applikation erhalten. Was mit `Create...()` erzeugt wurde, wird natürlich mit `DestroyWindow()` zerstört. Bei modalen Dialogen dagegen gehört zu `DialogBox()` am Ende `EndDialog()`.
  Nun ist ein nichtmodaler Dialog aber nur eine spezielle Sorte von Fenstern, wie sie mit `CreateWindow()` erzeugt werden können. Warum ist hier eine besondere Behandlung von Nachrichten per `IsDialogMessage()` nötig, statt dass diese das Dialog-Fenster per `DispatchMessage()` erreichen wie bei jedem anderen Fenster auch? Tatsächlich geschieht das auch mit den meisten Nachrichten über diesen Weg, auch bei nichtmodalen Dialogboxen. `IsDialogMessage()`

verarbeitet nur Tastatureingaben für den nichtmodalen Dialog! Damit wird eine Navigation per Tab, Return und Pfleiltasten in diesem Dialog (aber auch jedem aderen Fenster) möglich. Anscheinend kann `DispatchMessage()` selbst bei Tastaturnachrichten nicht erkennen, an welchen Dialog sie geschickt werden müssen, weshalb eine spezialisierte Behandlung nötig ist.

Aus dem MSDN zum Unterschied zwischen modalen und nicht-modalen Dialogboxen: »A modal dialog box requires the user to close the dialog box before activating another window in the application. However, the user can activate windows in different applications. A modeless dialog box does not require an immediate response from the user. It is similar to a main window containing controls.«

- Es gibt einen Unterschied im strukturellen Aufbau von `WindowProc()`- und `DialogProc()`-Callback-Funktionen.
  - Jede `WindowProc()`-Callback-Funktion gibt für jedes behandelte Ereignis einen ereignisspezifischen Rückgabewert zurück. Nicht selbst behandelte Ereignisse übergibt sie der `DefWindowProc()` und gibt dann deren Rückgabewert der Ereignisbehandlung zurück:

    ```
    switch(uMsg) {
      //...
      default:
        return DefWindowProc(hwnd, uMsg, wParam, lParam);
        break;
    }
    ```

  - Jede `DialogProc()`-Callback-Funktion ruft nicht selbst die `DefDlgProc()`-Funktion für nicht behandelte Ereignisse auf, sondern überlässt das der Nachrichtenbehandlungsroutine des Dialogs (z.B. in `DialogBox()`). Um ihr anzuzeigen, ob die erhaltene Nachricht bereits behandelt wurde, gibt sie für jede behandelte Nachricht `TRUE` zurück und für ansonsten `FALSE`, also am Funktionsende.
    Weitere Informationen aus dem MSDN: »Typically, the dialog box procedure should return `TRUE` if it processed the message, and `FALSE` if it did not. If the dialog box procedure returns `FALSE`, the dialog manager performs the default dialog operation in response to the message.
    If the dialog box procedure processes a message that requires a specific return value, the dialog box procedure should set the desired return value by calling `SetWindowLong(hwndDlg, DWL_MSGRESULT, lResult)` immediately before returning `TRUE`.«

- Deklarationen

  `char * p[10]` Ein Feld von 10 Elementen vom Typ `char *`.

  `char (*p)[10]` Ein Zeiger auf ein Feld von 10 Elementen vom Typ `char`.

- Wichtige ASCII-Codes:
  - »0«: 30h
  - »A«: 41h
  - »a«: 61h

- `WM_COMMAND`

  - `HIWORD(wParam)`: notification code, wenn die Nachricht von einem Control stammt. Etwa `EN_CHANGE`.
  - `LOWORD(wParam)`: ID des Menüeintrags, Controls oder Accelerators.

- Code um die Titelzeilen aller Fenster zu ermitteln:

```
char buf[5000];
char * p = buf;
for (HWND hNext = GetWindow(hwnd,GW_HWNDFIRST);
          hNext != NULL;
          hNext = GetWindow(hNext, GW_HWNDNEXT))
{
  if (p < buf+5000) {
    p[0] = "\n";
    p += GetWindowText(hNext, ++p, buf+5000-p);
  }
}
MessageBox(hwnd,buf,"titel",MB_OK);
```

4 Aufgabensammlung

Hier finden sich Codefragmente wie Funktionsaufrufe, Befehlssequenzen, aber auch kleine vollständige Beispiele, jeweils mit ausführlicher Dokumentation um zu verstehen, was die einzelnen Funktionen der Windows-API leisten. Die Beispiele basieren teilweise auf den Aufgabenstellungen alter Klausuren.

Die bisher aufgenommenen Aufgaben entsprechen den Teilaufgaben der Aufgaben 1-4 der Testklasur aus dem Sommersemester 2003. Diese ist identisch mit der Testklausur aus dem Sommersemester 2004.

4.1 WindowProc()

Der Name der benutzerdefinierten Callback-Funktion, die die an ein Fenster gesendeten Nachrichten verarbeitet, ist vom Benutzer frei wählbar. Im MSDN wird als Platzhalter `WindowProc()` verwendet, Programme verwenden häufig auch den Namen `WndProc()`. Die Funktion `WindowProc()` wird während der Nachrichtenbehandlung in der Haupt-Nachrichtenbehandlungsschleife eines Fensters durch `DispatchMessage()` aufgerufen. Dabei muss `WindowProc()` den folgenden Typ haben:

```
LRESULT CALLBACK WindowProc (HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
```

Der Benutzer beende das Programm durch Klick auf den Menüeintrag »Beenden«. Er wird identifziert durch `IDM_BEENDEN`. Welches Ereignis wird generiert und mit welchen Parametern wird dann `WindowProc()` aufgerufen? Nachdem das Fenster zerstört wurde, wird das Ereignis `WM_DESTROY` gesendet; in seiner Behandlung sollten Ressourcen freigegeben werden und dann `WM_QUIT` generiert werden durch Aufruf von `PostQuitMessage()`. Dieses letztere Ereignis führt dann zum tatsächlichen Beenden des aktuellen Threads (engl. *(to) quit*: beenden). Im Gegensatz zu allen anderen Ereignisses wird dieses nicht von `WindowProc()` behandelt, sondern von `GetMessage()` zu Beginn der Nachrichtenschleife. Damit endet das Programm vor dem Aufruf von `WindowProc()` mit dem Ereignis `WM_QUIT`!

Für das Ereignis `WM_DESTROY` wird `WindowProc()` mit folgenden Parameterwerten aufgerufen:

`hwnd` Enthält das Handle des Hauptfensters.

`uMsg` `WM_DESTROY`

`wParam` <unused>

`lParam` <unused>

4.2 User-Extra-Bytes in WNDCLASS

Die `WNDCLASS`-Struktur enthält die Fensterklassen-Attribute, die mit der Funktion `RegisterClass()` registrert wurden. Eine `WNDCLASS`-Instanz definiert eine (logische) Fensterklasse, deren (logische) Instanzen einzelne Fenster sind. So werden »Klassen« und »Objekte« in C simuliert. Die Klasse enthält dabei die Daten, die von mehreren Fenstern einer Applikation gemeinsam benötigt werden. Einzelne Fenster, also Instanzen zu einer existierenden `WNDCLASS`-Struktur, legt man mit `CreateWindow()` an. Auf die in der Fensterklasse enthaltenen Informationen kann man mit `GetClassInfo()` zugreifen.

Folgende Elemente einer `WNDCLASS`-Struktur können dabei User-Extra-Bytes enthalten:

`int cbClsExtra` Die Anzahl der gewünschten zusätzlichen Bytes nach der `WNDCLASS` structure. Das System alloziert sie und initalisiert sie mit 0.

int cbWndExtra Die Anzahl der gewünschten zusätzlichen Bytes nach einer Fensterinstanz dieser `WNDCLASS`. Das System alloziert sie und initalisiert sie mit 0.

Während MSDN für den zusätzlichen Speicherplatz in beiden Fällen keine Größenbeschränkung nennt und sich dabei auf Windows 95 und höher bezieht, gilt für Windows 3.11 in beiden Fällen anscheinend eine Beschränkung auf höchstens 39 Byte.

4.3 DefWindowProc()

Während `CallWindowProc()` eine benutzerdefinierte Nachrichtenbehandlungsroutine aufruft, ruft `DefWindowProc()` die Default-Nachrichtenbehandlungsroutine auf. `DefWindowProc()` wird üblicherweise nach jeder benutzerdefinierten Nachrichtenbehandlung aufgerufen und garantiert, dass alle übriggebliebenen Nachrichten ebenfalls behandelt werden. Die Parameter entsprechen denen einer benutzerdefinierten Nachrichtenbehandlungsroutine (üblicherweise mit dem Pseudonym `WindowProc()` bezeichnet):

```
LRESULT DefWindowProc( HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam );
```

`hWnd` Handle des Fensters, das die Nachricht empfangen hat.

`Msg` Spezifiziert die Nachricht.

`wParam` Zusätzliche, nachrichtenspezifische Information.

`lParam` Zusätzliche, nachrichtenspezifische Information.

4.4 Initialisierung von char-Strings

»Für char-Strings gelte Byte-Alignment und die folgenden Initialisierungen:

```
char szStr[][3] = { "A", "B", "C", "" };
char * p = "A\000B\000C\000";
char buf[] = "A\0B\0C\0";
```

Geben Sie die belegten Bytes im Speicher in hexadezimaler Notation an:« Wir nehmen dabei an, dass es sich um lokale Variablen handelt. Sie werden nicht vom System initialisiert, weshalb wir an solchen Stellen für die unbekannten Werte `0x__` schreiben.

```
//für szStr ( aufsteigende Adressen ):
0x41 0x00 0x__ 0x42 0x00 0x__ 0x43 0x00 0x__ 0x00 0x__ 0x__ 0x__
//für * p ( aufsteigende Adressen ):
0x41 0x00 0x42 0x00 0x43 0x00 0x00 0x__ 0x__ 0x__ 0x__ 0x__ 0x__
//für buf ( aufsteigende Adressen ):
0x41 0x00 0x42 0x00 0x43 0x00 0x00 0x__ 0x__ 0x__ 0x__ 0x__ 0x__
```

4.5 MessageBox und wsprintf()

»Mit `wsprintf()` können C-ASCII-Strings formatiert in einen Buffer geschrieben werden.

```
int wsprintf(
  LPTSTR lpOut,  // pointer to buffer for output
  LPCTSTR lpFmt, // pointer to format-control string
  ...            // optional arguments
);
```

Wie sieht die `MessageBox()`-Ausgabe für das folgende Beispiel aus?

```
union _STR { //überdeckender Speicher
  char * str[ ];
  char buf[256];
};
_STR s = { "Dies.ist\n...Str1", "Dies.ist" ".Str2" };
wsprintf(s.buf, "\n1: %s\n2: %s", s.str[0],s.str[1]);
MessageBox(0, s.buf, "s.Buf", 0);
```

Wie sieht die spaltentreue Anzeige der MessageBox etwa aus?

```
|=========================================|
|  |s|.|B|u|f| | | | | | | | | | | | | | | | |-----------------------------------------
|                                           |
| 1|:| |D|i|e|s|.|i|s|t| | | | | | | | | | |
| .|.|.|S|t|r|1| | | | | | | | | | | | | | |
|                                           |
| 2|:| |D|i|e|s|.|i|s|t| | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | |
|                                           |
|                                           |
|                 ---------                 |
|                |  |O|K|  |                |
|                 ---------                 | |=========================================
```

Bemerkungen:

- Die Lösung wurde noch nicht am Rechner getestet.
- Die MessageBox hat als einzigen Button »OK«, weil dies die Standardeinstellung ist. Diese wird hier genutzt, weil diese MessageBox keine Konstante wie `MB_OKCANCEL`, `MB_OK` o.ä. als viertes Argument übergeben bekam. Vgl. MSDN :: MessageBox Function http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/dialogboxes/dialogboxreference/dialogboxfunctions/messagebox.asp.
- Werden Unions statisch initialisiert, so wird nur ihr erstes Element initialisiert. In diesem Fall ist das das Array von C-Strings `char * str[ ]`.
- `s.str[2]` wird initialisiert mit `"Dies.ist" ".Str2"` ⇔ `"Dies.ist\0.Str2"`, denn jedes String-Literal (d.i. jede zwischen Anführungszeichen eingeschlossene Zeichenkette) in C bedeutet einen Null-terminierten String. Deshalb endet die Ausgabe bei der ersten Null-Terminierung.

4.6 Das Makro `err_if()` vervollständigen

»Das folgende Macro soll zur vollen Funktionsfähigkeit vervollständigt werden. `__FILE__` setzt »automatisch« den Pfad-File-Namen ein. Bei `#define NO_DEBUG` anstelle von `#undef NO_DEBUG` soll eine Neu-Übersetzung KEINEN Maschinen-Code für die `err_if()`-Aufrufe generieren, obwohl die `err_if()`-Aufrufe im Quelltext verbleibenden sollen.

```
#undef NO_DEBUG //bzw. #define NO_DEBUG
#ifndef NO_DEBUG
#define err_if(exp) \
  if(exp){\
    char buf[512];\
    wsprintf(\
      buf,\
      "File\t: %s:\nZeile\t: %d:\nFehler\t: %s:",\
      __FILE__,\
      __LINE__,\
      #exp\
    );\
    if(MessageBox(NULL,buf,0,MB_OKCANCEL)-1)\
      *((void **)0)=0;\
  }
#else #define err_if(exp) /* nothing */
#endif
```

4.7 MessageBox bei err_if(exp)

»Im Quelltext komme der Macro-Aufruf err_if():

```
double * p =new double[1024];
err_if( p == NULL );
```

vor. Dann sieht die `MessageBox()`-Ausgabe (im Fehlerfall bei `p==NULL`) etwa wie folgt aus (bitte ergänzen):« Dateiname und Zeilennummer müssen hier angenomen werden. Nach dem Präprozessorlauf sieht der Quellcode wie folgt aus:

```
double * p =new double[1024];
if( p == NULL ){
  char buf[512];
  wsprintf(
    buf,
    "File\t: %s:\nZeile\t: %d:\nFehler\t: %s:",
    "c:\Eigene Dateien\helloworld.cpp",
    168,
    " p == NULL "
  );
  if(MessageBox(NULL,buf,0,MB_OKCANCEL)-1)
    *((void **)0)=0;
}
```

Dann sieht die MessageBox wie folgt aus:

```
|=============================================|
|                                             |
|---------------------------------------------|
| File    : c:\Eigene Dateien\helloworld.cpp  |
|                                             |
| Zeile   : 168                               |
|                                             |
| Fehler  :  p == NULL                        |
|           ----------    ----------          |
|           |   OK   |    | CANCEL |          |
|           ----------    ----------          |
|=============================================|
```

4.8 Alternative zu err_if()

»Wie ändert sich die Ausgabe, wenn anstelle von `*((void **)0)=0;` die Alternative `_asm { int 3 };` gewählt wird? Bemerkung: Beim Pentium Processors bewirkt die "`int 3`"-Assembler-Inline-Instruction einen "Hardware-Breakpoint".«

Der Breakpoint wird erst ausgelöst, nachdem der Benutzer auf die MessageBox reagiert hat. Die Anzeige der MessageBox ändert sich deshalb nicht. Bei beiden Alternativen wird ein Laufzeitfehler ausgelöst, wenn der `if`-Block nach Anzeige der MessageBox ausgeführt wird. Das geschieht stets, außer die MessageBox gibt `1` zurück. Weil in einem Windows-Header-File `#define IDOK 1` definiert ist, kann durch den OK-Button ein Fehler anerkannt werden, ohne das Programm zu beenden. Durch den Cancel-Button wird das Programm beendet.

»Welches bekannte C/C++-Macro entspricht etwa dem obigen `err_if(exp)`-Macro? Macro-Name:« Gute Frage. Vielleicht ist die Funktion `assert()` gemeint?

4.9 CenterWindow() ergänzen

»Es ist der Quelltext für eine Funktion `CenterWindow(hwndChild, hwndParent)` zum Zentrieren eines Child-Window im Parent-Window zu ergänzen. Hinweise:

- `typedef struct {LONG left;LONG top;LONG right;LONG bottom;} RECT;` structure defines the coordinates of the upper-left and lower-right corners
- `BOOL GetWindowRect(HWND hWnd,LPRECT lpRect);` retrieves the dimensions of the window-bounding rectangle
- `BOOL GetClientRect(HWND hWnd,LPRECT lpRect);` retrieves the coordinates of a window's client area.
- Die Funktion `int GetSystemMetrics(int nIndex);` gibt gerätespezifische Werte zurück. Z.B. kann `nIndex` sein:

  `SM_CXBORDER` width in pixels, of a window border.

  `SM_CYBORDER` height in pixels, of a window border.

  `SM_CXFULLSCREEN` width of the client area for a full-screen-window

  `SM_CYFULLSCREEN` height of the client area for a full-screen-window

  `SM_CXSCREEN` width in pixels of the screen

  `SM_CYSCREEN` height in pixels of the screen

Ergänzen Sie die folgende Funktion `CenterWindow()`, die das Child-Window `hwndChild` im Parent-Window `hwndParent` mittig positioniert.

```
BOOL CenterWindow (HWND hwndChild, HWND hwndParent) {
  int  xNew, yNew;
  RECT rChild, rParent, rScreen;
  int  wChild, wParent; // Breiten
  int  hChild, hParent; // Hoehen
  //Get_____________Rect(hwnd________, _____________);
  GetWindowRect(hwnd hwndChild, (LPRECT) & rChild);
  wChild = rChild.right  - rChild.left;
  hChild = rChild.bottom - rChild.top;
  //Get_____________Rect(hwnd________, _____________);
  GetWindowRect(hwnd hwndParent, (LPRECT) & rParent);
  wParent = rParent.right  - rParent.left;
  hParent = rParent.bottom - rParent.top;
  rScreen.left   = rScreen.top = 0;
  rScreen.right  = GetSystemMetrics(SM_CXSCREEN);
  rScreen.bottom = GetSystemMetrics(SM_CYSCREEN);
  xNew = rParent.left + ((wParent - wChild)/2);
  yNew = rParent.top  + ((hParent - hChild)/2);
  if (xNew < rScreen.left)
    xNew = rScreen.left;
  else
    if ((xNew + wChild) > rScreen.right)
      xNew = rScreen.right - wChild;
  if (yNew < rScreen.top)
    yNew = rScreen.top;
  else
    if ((yNew + hChild) > rScreen.bottom)
```

```
      yNew = rScreen.bottom - hChild;
  return SetWindowPos ( hwndChild, NULL,
    xNew, yNew, 0,0, SWP_NOSIZE|SWP_NOZORDER);
}
```

4.10 About-Dialog

Die Dialog-Resource eines modalen About-Dialoges habe den Identifizierer `IDD_ABOUT` und die CALLBACK-Funktion `dlgAbout()`. Der Dialog habe eine Titelzeile (`WS_CAPTION`) mit einem SystemMenue (`WS_SYSMENU`) und enthalte die Buttons `IDOK` und `IDCANCEL`.

```
//aus Windows-Header-Files:
#define MB_OK 0x00000000L
#define IDOK 1
#define WM_CLOSE 0x0010
#define SC_CLOSE 0xF060
```

Bitte ergänzen Sie `dlgAbout()` bei den markierten Stellen.

bei a) Beim Beenden des modalen Dialoges mit `WM_CLOSE` soll an die aufrufende Funktion `DialogBox()` der Rückgabewert `WM_CLOSE` zurück geben werden.

bei b) Beim Beenden des Dialoges mit einem Klick auf den `IDOK`-Button soll `DialogBox()` den Wert von `IDOK` zurück geben. Beim Beenden des Dialoges mit einem Klick auf den `IDCANCEL`-Button soll eine `MessageBox()` den numerischen Wert von `IDCANCEL` anzeigen und auch `IDCANCEL` zurück geben.

bei c) Beim Beenden des Dialoges mit einem SystemMenü-Klick soll `DialogBox()` den `wParam`-Wert `SC_CLOSE` zurück geben.

bei d) Was ist unter `WM_DESTROY` einzutragen, damit die Haupt-Nachrichten-Schleife `while (GetMessage(& msg,0,0,0)) {/*...*/}` *nicht* beendet wird?

```
BOOL CALLBACK dlgAbout
(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
  switch ( iMsg ){ //(a)
    case WM_CLOSE: EndDialog(hwnd, WM_CLOSE);
      return TRUE;
    case WM_COMMAND:{
      switch ( LOWORD(wParam) ){ //(b)
        case IDOK: EndDialog(hwnd, IDOK);
          return TRUE;
        case IDCANCEL:
          EndDialog(hwnd,
            MessageBox(hwnd,itoa(IDCANCEL),0,MB_CANCEL)
          );
          return TRUE;
        break ;
      }
      } break;//ende WM_COMMAND
```

```
    case WM_SYSCOMMAND: {//(c)
      switch ( wParam ) { //wParam anders verwendet als bei WM_COMMAND
        case SC_CLOSE:
          EndDialog(hwnd, wParam);
          return TRUE;
        }
        } break; //ende WM_SYSCOMMAND
             //(d)
    case WM_DESTROY:
      return TRUE;
      break;
  } // ende switch ( iMsg )
  return FALSE; //(d)
}
```

4.11 DialogBox()-Aufruf

bei e) In (dem folgenden Ausschnitt) der `WndProc()` des Hauptfensters soll unter dem vorhandenen Menü-Punkt `IDM_MODAL` der `DialogBox()`-Aufruf eingetragen werden. Hinweis:

```
int DialogBox(
  HINSTANCE hInstance, //handle to application instance
  LPCTSTR lpTemplate,  //identifies dialog box template
  HWND hWndParent,     //handle to owner window
  DLGPROC lpDialogFunc //pointer to dialog box procedure
);
```

bei f) Unmittelbar nach e) (d.h. nach `DialogBox()`) soll in einer `MessageBox()` der zugeordnete String (`"IDOK"` bzw. `"IDCANCEL"` bzw. `"WM_CLOSE"` bzw. `"SC_CLOSE"`) angezeigt werden.

```
LRESULT CALLBACK WndProc
(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam) {
   switch(iMsg) {
     case WM_COMMAND :
       switch ( LOWORD(wParam) ) {
         case IDM_MODAL: //(e)
           int res = DialogBox(
             GetModuleHandle(0),
             MAKEINTRESOURCE(IDD_ABOUT),
             hwnd,
             (DLGPROC) dlgAbout
           );
           switch(res){ //(f)
             case IDOK:     MessageBox(0,"IDOK",0,0);    break;
             case IDCANCEL: MessageBox(0,"IDCANCEL",0,0);break;
             case WM_CLOSE: MessageBox(0,"WM_CLOSE",0,0);break;
             case SC_CLOSE: MessageBox(0,"SC_CLOSE",0,0);break;
           }
           break; //ende IDM_MODAL
       } //switch ( LOWORD(wParam) )
```

```
    } //switch(iMsg)
    return DefWindowProc(hwnd,iMsg,wParam,lParam);
}
```

4.12 Ein Programm beenden

Das Beenden eines Fensters oder eine Applikation ist dreistufig:

1. `WM_CLOSE`: Der Benutzer oder ein anderes Fenster wünscht, das Programm zu beenden. In der Behandlung dieser Nachricht wird ggf. eine Rückfrage gestellt und dann `WM_DESTROY` gesendet, indem `DestroyWindow()` aufgerufen wird.
2. `WM_DESTROY`: Diese Nachricht zeigt an, dass nun das Fenster zerstört werden soll. In der Behandlung dieser Nachricht wird `WM_QUIT` gesendet durch `PostQuitMessage(0)`.
3. `WM_QUIT`: Diese Nachricht wird nicht mehr von der Callback-Funktion behandelt, sondern bricht die Nachrichtenschleife schon zu Beginn ab. Dadurch endet das Programm.

Literatur

[1] Skript von Prof. Dr. Bachmann. Zu finden auf seiner Homepage http://homepages.fh-giessen.de/~hg54. Es wird aktualisiert, ist jedoch relativ stabil. Es macht eine eigene Mitschrift unnötig. Am Semesterende wird es auch als PDF-Datei zur Verfügung gestellt. Vorteil der Vorlesung gegenüber diesem Skript sind die anschaulicheren Erklärungen.

[2] Homepage von Prof. Dr. Christidis. Dort ist das Material von Prof. Dr. Christidis zu dieser Veranstaltung vorhanden. Alles was in der Vorlesung präsentiert wird, kann man hier als PDF herunterladen. Hier sind nicht alle in der Vorlesung gezeigten Folien enthalten, sondern nur die, die man zum Lernen braucht. http://homepages.fh-giessen.de/~hg11237

[3] ALABAMA. Auf diesem Server befindet sich Material von Prof. Christidis aus dem WS 2002/2003. Man kann es nutzen, um für die Veranstaltung zu lernen und die Klausur bei Prof. Christidis (auch gut) zu bestehen.

[4] Regionales Rechenzentrum für Niedersachsen (RRZN): / Softwareberatung Herdt: »DieProgrammiersprache C. Ein Nachschlagewerk«, RRZN 1995. http://www.rrzn.uni-hannover.de/Dokumentation/Handbuecher/index.html. In dieser Serie gibt es etwa 200 Bücher zur Informatik, die nie mehr als 5 EUR kosten. Bezugsquelle: HRZ der JLU Gießen, Heinrich-Buff-Ring 44, 35392 Gießen.

[5] B. W. Kerningham, D. M. Ritchie: »Programmieren in C«. (2. Ausgabe, ANSI-C), Carl Hanser 1990. Das Standardwerk zu C, von den Erfindern von C.

[6] H. Schildt: »C - The Complete Reference«; McGraw-Hill 2000. Das Lieblingsbuch von Prof. Dr. Christidis zu C. Bald in englischer und deutscher Fassung in der Bibliothek vorhanden.

[7] M. A. Ellis, B. Stroustrup: »The Annotated C++ Reference Manual« (ANSI-Base Document), Addison-Wesley 1991.

[8] Ph. A. Laplante: »Real-Time Systems - Design and Analysis«; IEEE Press 1993. Ein sehr gründliches Buch zu Echtzeitsystemen. Nicht nötig für diese Veranstaltung, aber eine Empfehlung bei der Beschäftigung mit Echtzeitsystemen.

[9] J. Nehmer, P. Sturm: »Systemsoftware - Grundlagen moderner Betriebssysteme« dpunkt-Verlag 1998.

[10] P. Pepper: »Grundlagen der Informatik« (2. Auflage), R. Oldenburg 1995.

[11] Andrew S. Tanenbaum: »Modern Operating Systems« (2nd Edition) Prentice Hall 2001. Die deutsche Übersetzung ist gut übersetzt, hat aber noch viele Fehler, die bei der Übersetzung hereingekommen sind.

[12] Andrew S. Tanenbaum: »Betriebssysteme - Entwurf und Programmierung«

[13] Ch. Petzold: »Windows Programmierung« (5. Auflage) Microsoft Press 2000. Das Standardwerk für Windows-Programmierung. Es fängt mit den einfachsten Dingen an. Preis etwa 60 EUR. Die Übersetzung ist sehr gut. Auf der CD ist das englischsprachige Original vorhanden. Das Buch ist auch in der Bibliothek der FH Gießen-Friedberg vorhanden.

[14] D. A. Solomon, M. Russinovich: »Inside Microsoft Windows 2000«; (3. Auflage) Microsoft Press 2000