Software-Interrupts in C

Franz Fiala ftp://pcnews.at/pcn/67/fiala/interrupts/

BIOS

Das BIOS (**B**asic **I**nput **O**utput **S**ystem) eines Rechners ist ein mit dem Rechner mitgeliefertes Programmstück als Bindeglied zwischen der Hardware und dem danach folgenden Betriebssystem.

Jemand, der ein einem BIOS entsprechendes Programmstück programmiert, programmiert in den Registern der Hardware, wir nannten es *Hardwareprogrammierung.* Die Gründe dafür sind:

- BIOS-Funktion nicht vorhanden,
- BIOS zu langsam,
- BIOS unvollständig,
- um zu lernen.

Jedenfalls gilt die Regel, dass man das BIOS benutzen soll, wo immer es geht.

Bootvorgang

Das BIOS ist (ähnlich wie auch das Betriebssystem) eine Sammlung von Programmen, die ein Programmierer nutzen kann. Es hat aber auch ein Hauptprogramm, das Boot-Programm, mit dem alle Rechnerkomponenten initialisiert und überprüft werden und abschließend der Bootsektors geladen und mit dessen Bootprogramm fortgesetzt wird.

Das Boot-Programm wird beim Einschalten des Rechners abgearbeitet. Der Startpunkt für dieses Programm ist die Adresse 0f000:fff0, der Reset-Vektor für den Kalt- oder Warmstart. Während dieses Startprogramms wird das Testprogramm POST durchlaufen.

POST

Der Anwender eines PC merkt normalerweise kaum etwas von diesem Programmteil. Hardwareprofis können aber aus den Fehlermeldungen von POST eine Menge diagnostische Hinweise erhalten. POST hat drei Ausgabemöglichkeiten: Bildschirm (wenn Bildschirmkarte funktioniert), Lautsprecher (wenn dieser ansprechbar ist) oder Port 80H (wenn sonst keine sonstige Ausgabe möglich ist). Allerdings bekommt man aus den Bitmustern an Port 80h viele Fehler mitgeteilt, die aber leider bei jedem BIOS anders sind. Herbert Sommerer berichtete darüber in der Ausgabe PCNEWS 56.

Die letzte Aktivität des BIOS-Start-Programms ist die Entscheidung, was als Nächstes zu geschehen hat. Gemäß der Boot-Reihenfolge wird der Boot-Sektor entweder von der Diskettenstation `A:` oder `C:` geladen. Booten von Laufwerk `A:` kann ausgeschaltet werden, was Einschleppen von Viren durch versehentliches Booten von `A:` vermeidet.

Zu diesem Zeitpunkt wurden durch das BIOS die Interruptvektoren initialisiert oder enthalten zumindest einen Vektor, der auf einen leeren IRET-Befehl zeigt. (siehe Tabelle weiter hinten). Wenn das BIOS in einem Speicherbereich eine BIOS-Extension entdeckt (z.B. Netzwerkkarte oder Videokarte), wird dessen Initialisierungsroutine angesprungen.

Betriebssystem

Das Betriebssystem eines Rechners stellt folgende Dienstleistungen zur Verfügung:

- dem Benutzer einen Kommandosatz zur Verwaltung seines Rechners; (z.B. DOS-Befehle durch `COMMAND.COM`)
- dem Programmierer genau definierte Schnittstellen, um diese Dienste in Anspruch nehmen zu können. (Int 20h, 21h,..2fh)

Es ist also wieder wie das BIOS eine Sammlung von Programmen. Es hat aber auch ein Hauptprogramm, den Kommandointerpreter, der geladen wird, wenn das Laden des Betriebssystems und der Treiber abgeschlossen ist. Beim MSDOS heißt der Kommandointerpreter `COMMAND.COM`. Verschiedene Public-Domain- und Shareware-Autoren bemühten sich, verbesserte Versionen von `COMMAND.COM` zu verfassen. Will man einen anderen Kommandointerpreter benutzen, muss man eine entsprechende Eintragung in der Datei `CONFIG.SYS` beim Eintrag `SHELL=` vornehmen.

Das Betriebssystem MSDOS initialisiert jetzt weitere Interruptvektoren und ersetzt oder erweitert bestehende BIOS-Vektoren (versteckte Systemdatei `IO.SYS`) und fügt seine eigenen Interrupts 20..2f dazu (versteckte Systemdatei `MSDOS.SYS`).

Da aber das Betriebssystem während des Bootvorgangs auch verschiedene Treiber lädt, erweitert es die ursprüngliche Funktionalität. Wichtigste Erweiterungen sind

- Maustreiber (Interrupt 33h),
- Netzwerktreiber, ...

Sie alle belegen weitere Interrupts.

C und Interrupts

Die Sprache C hat einerseits den maschinenunabhängigen Kern ANSI-C und darüber hinaus Erweiterungen, um eine Verbindung zwischen der aktuellen Maschine (PC) und dem aktuellen Betriebssystem (DOS; Windows, NT, OS/2, UNIX) herzustellen.

Diese Schnittstellen sind

- Einsprungadressen,
- Adressbereiche mit definierten Inhalten,
- Registerinhalte,
- Interruptvektoren.

Maschinenunabhängige Programme erfordern, dass Betriebsystemdienste auf denselben Adressen zu finden sind oder jedenfalls auf eindeutig auffindbaren Adressen. Das Programm benutzt dann diese Einsprungpunkte und übergibt dem Betriebssystem einen Satz von Parametern, damit dieses die Aufgabe übernehmen kann.

Im Betriebssystem CP/M, dem Vorgänger von MSDOS, benutzte man zum Aufruf des Betriebssystems einen Unterprogrammaufruf auf Adresse 5. In CP/M gab es als gemeinsamen Nenner aller Rechner lediglich das Betriebssystem. Zwar gab es eine gemeinsame CPU, aber keinen gemeinsamen Hardwareaufbau, der über ein einheitliches BIOS erreichbar gewesen wäre. Ein BIOS-Call war in diesen Zeiten von vornherein nicht gestattet, da es kein portables Programm gewesen wären.

In den ersten Tagen von MSDOS gab es auch Rechner mit verschiedenartiger Hardware (und daher mit verschiedenem BIOS), diese Rechner sind aber praktisch alle vom Industriestandard-PC verdrängt worden.

In der CPU 80x86 sind 256 vektorisierte Interrupts implementiert, die man auch in Programmen verwenden kann. Der Assemblerbefehl dazu lautet

```
INT  n
```

wobei `n` eine von 256 Nummern ist. Ursprünglich hatten Interrupts eigentlich den Zweck, asynchron zum Programm eintreffende Hardwareereignisse der CPU zu melden. Dazu liefert der CPU ein externer Interrupt-Controller den einfachen Befehl `INT n` mit der richtigen Nummer und die CPU führt die richtige Interrupt-Serviceroutine aus.

Der PC braucht aber nur 16 Hardware-Interrupts, der Rest ist frei. Die restlichen Befehle `INT n` stehen also für andere

Zwecke zur Verfügung. Die Festlegung am MSDOS-PC war folgende:

Interrupt	Aufgabe
00..07	CPU-Interrupts
08..0f	Hardware-Interrupts (0..7)
10..1f	BIOS-Interrupts
20..2f	MSDOS
a0..a7	Hardware-Interrupts (8..15)

Im Laufe der Enwicklung rund um den PC wurden die noch verfügbaren Interrupts rasch durch neu dazukommende Dienste belegt. Eine genaue Aufstellung entnehmen Sie der Tabelle weiter hinten.

Die große Zahl verschiedener Interrupts und der zugehörigen Detailfunktionen macht eine systematische Besprechung für eine Zeitschrift schwierig. Es ist bei Verständnis des Prinzips auch gar nicht nötig, für jeden einzelnen Interrupt ein Beispiel zu besitzen.

Eine Schwierigkeit bei der Besprechung ist, dass nicht alle Interrupts von einem einzigen Hersteller definiert wurden. Entsprechend müssen Interrupttabellen aus verschiedenen Quellen zusammengestellt werden.

Die ursprünglichen Interrupts kommen von IBM, Version AT-286. Diese BIOS-Interrupts werden von den Erzeugern der verschiedenen BIOS-Varianten am jeweils aktuellen Stand gehalten. Nicht jeder Erzeuger lässt sich aber in die Karten schauen. Einer, der sich traut, ist Phoenix. Es gibt eine komplette Liste aller BIOS-Interrupts des Phoenix-Bios:

System BIOS for IBM PCs, Compatibles and EISA-Computers, Second Edition, Phoenix Technical Reference Series

Zusammenfassungen quer über alle weiteren Funktionen der Interrupts, wie z.B. MSDOS, Maus, Netzwerke, Bildschirmkartenerweiterungen, FOSSIL-Treiber usw., die alle nicht zum ursprünglichen BIOS gehören, werden erst im Laufe des Bootvorgangs geladen. Verzeichnisse über diese Interrupts sind beispielsweise:

- The Undocumented PC, Frank van Gilluwe
- The Undocumented DOS, Andrew Schulman

Da jeder dieser Interrupts vielfältige Aufgaben erfüllen kann, sind genaue Festlegungen der Kommunikation mit dem Interrupt erforderlich. Das wichtigste Kommunikationsmittel sind die Register der CPU.

Register

Die wichtigste Form der Parameterübergabe ist der Registerinhalt. Daher ist es notwendig, den Registeraufbau unserer CPU genau zu kennen. Die 80x86-CPU besitzt die Register `AX`, `BX`, `CX` und `DX` für den allgemeinen Bedarf und für Adressieroperationen auch noch die Register `SI`, `DI`. Diese Register sind 16-bit-Register. Zur Bildung einer vollständigen 20-bit-Adresse werden darüber hinaus auch noch die Segmentregister `CS`, `DS`, `ES` und `SS` benötigt.

Unsere CPU kann mit 8-bit-Daten genauso gut umgehen wie mit 16-bit-Daten. Die Arbeitsregister `AX`, `BX`, `CX` und `DX` sind daher in eine jeweils niederwertige Hälfte `AL`, `BL`, `CL` und `DL` und in eine höherwertige Hälfte `AH`, `BH`, `CH` und `DH` geteilt.

Zu diesen Registern kommen noch eine Reihe von Flags (Kennzeichenbits), die den aktuellen Zustand der CPU oder der vorangegangenen Operation kennzeichnen. Aus praktischen Gründen werden diese Bits in einem gemeinsamen 16-bit-Wort zusammengefasst. Die Bedeutung der Bits im einzelnen ist:

```
1111ODIT SZ?A?P?C
```

Z	Zero
S	Sign
T	Trap
C	Carry
I	Interrupt
P	Parity
D	Direction
O	Overflow
A	Auxillary Carry

Register einer CPU sind nicht Bestandteil der Sprachdefinition einer Hochsprache. Wenn man daher eine Hochsprache wie C für die Kommunikation mit CPU-Interrupts verwenden will, muss die betreffende Sprachvariante eigene Definitionen für entsprechenden C-Variablen, die genau das bewirken, geschaffen werden. Bei den MSDOS-C-Compilern werden folgende Strukturen in der Header-Datei `dos.h` verwendet:

```
/* Auszug aus dos.h */

struct _WORDREGS {
    unsigned int ax;
    unsigned int bx;
    unsigned int cx;
    unsigned int dx;
    unsigned int si;
    unsigned int di;
    unsigned int cflag;
    };

/* byte registers */
struct  BYTEREGS {
    unsigned char al, ah;
    unsigned char bl, bh;
    unsigned char cl, ch;
    unsigned char dl, dh;
    };

/* general purpose registers union -
 *  overlays the corresponding word and byte
registers.
 */
union _REGS {
    struct _WORDREGS x;
    struct  BYTEREGS h;
    };

/* segment registers */
struct  SREGS {
    unsigned int es;
    unsigned int cs;
    unsigned int ss;
    unsigned int ds;
    };
```

Dieser Auszug stammt aus dem Compiler VISUAL C++, Datei `dos.h`. Man sieht, dass die früheren Registerbezeichnungen noch ohne Unterstrich, die ANSI-konformen, die durch das Makro `__STDC__` unterschieden werden, mit einem Unterstrich beginnen.

Aufruf der Interrupts in C

Um einen bestimmten Interrupt in C aufrufen zu können, findet man in `dos.h` folgende Funktionsprototypen:

```
int _intdos(union _REGS *rin, union _REGS
*rout);
int _intdosx(union _REGS *rin, union _REGS
*rout, struct  SREGS *sreg);
int _int86(int, union _REGS *rin, union
 REGS *rout);
int _int86x(int, union _REGS *rin, union
_REGS *rout, struct _SREGS *sreg);
```

Ein beliebiger Interrupt ist mit `_int86()` und `_int86x()` aufrufbar, für den sehr häufig verwendeten MSDOS-Interrupt 0x21 gibt es auch die Kurzformen `_intdos()` und `_intdosx()`.

Jeder dieser Interrupts bekommt zwei Parameter `rin` und `rout` in Form eines Pointers auf einen Registersatz übermittelt. `rin` enthält die Registerwerte vor Aufruf des Interrupts und `rout` enthält die Werte, die der betreffende Interrupt zurückliefert. In den meisten Fällen kann `rin` und `rout` derselbe Registersatz sein.

Es gibt Interrupts, die auch Segmentregister übernehmen oder übergeben. Für diese Interrupts sind die Versionen `_intdosx()` und `_int86x()` vorgesehen, die einen weiteren Parameter in Form eines Pointers auf eine Segmentregisterstruktur übernehmen und auch zurückliefern.

Ein einfaches Beispiel für die Anwendung eines BIOS-Interrupt zeigt das folgende Beispiel. Es soll festgestellt werden, wieviel Speicher zur Verfügung steht. In der Tabelle der BIOS-Interrupts finden wir, dass der Interrupt 0x12 diese Aufgabe erledigen kann und die Speichergröße in kByte im Register `AX` zurückliefert.

```
union REGS r;
int86(0x12,&r,&r);
printf("Der Speicher ist %u kByte groß\n",
  r.x.ax);
```

Zunächst muss eine Register-Variable definiert werden, die der Interrupt-Funktion `int86()` übergeben werden kann. Da nur ein Pointer und nicht die Struktur selbst

übergeben wird, verwendet man beim Aufruf den Adressoperator `&`.

BIOS- und MSDOS-Interrupts in der C-Bibliothek

Im Prinzip können alle BIOS- und MSDOS-Funktionen auf diese Interrupts zurückgeführt werden. Für sehr häufig vorkommende Anwendungen gibt es aber in der Standardbibliothek von C, beschrieben in den Dateien `bios.h` und `dos.h`, sehr viele konkretisierte Anwendungen. Die Funktionsgruppe des seriellen Interrupt wurde in den **PCNEWS**-42 beschrieben.

Nachteile von C

Auf diese Weise können auch alle anderen Interrupts in C verwendet werden. Es muss aber betont werden, dass bei Programmteilen, die geschwindigkeitsoptimiert werden müssen, vielleicht ein Bisschen anders programmiert werden sollte, da durch die Art des Aufrufs der Interrupt langsamer ausgeführt wird als z.B. eine Assembler-Routine, da immer alle Register übergeben werden und durch die Funktionen `int86()`.. auch immer alle in die jeweiligen Register geladen werden, auch, wenn sie gar nicht benötigt werden.

BIOS-Interrupts, HC08IB0.C

Das folgende Beispiel ist eine Demonstration, wie die vielfältigen BIOS-Interrupts verwendet werden können. Es werden wichtige Maschinendaten ermittelt und dargestellt. In einer ähnlichen Form erledigen das viele Utilities, aber auch jedes Programm, das sich in einer Umgebung zu orientieren hat, muss sie anwenden. Der Text, den das Programm ausgibt ist im nebenstehenden Kasten dargestellt.

Das Programm wurde sowohl in einer C- als auch in einer C++-Version geschrieben. Dargestellt ist der Anfang der C-Variante. Das vollständige Programm und die C++-Version finden Sie im Web.

Testen einiger BIOS-Interrupts

```
Speicherkapazität feststellen, Interrupt 0x12
Der Speicher ist 640 kByte groß
Ausstattung feststellen, Interrupt 0x11
Bitkombination gelesen :             c863 HEX
Bitkombination gelesen : 1100100001100011 BIN
Bitkombination         : ddsgsssDffvvrr7f
Laufwerk               : 1
Arithmetik-Prozessor   : nein
RAM auf Hauptplatine   : 0 x16k
Bildschirmmodus        : 80x25 Farbe
Anzahl der Laufwerke   : 1
DMA möglich            : ja
Anzahl der RS232-Karten: 4
Game-Port vorhanden    : ja
Serieller Drucker      : ja
Druckeranzahl          : 12

Laufwerke feststellen
Laufwerk 0: Laufwerkstyp :Floppy mit Diskettenwechselsignal
Laufwerk 1: Laufwerkstyp :Floppy mit Diskettenwechselsignal
Laufwerk 2: Laufwerkstyp :Festplatte
Anzahl der 512-byte-Sektoren:           510000
Laufwerk 3: Laufwerkstyp :Festplatte
Anzahl der 512-byte-Sektoren:           823344
Laufwerk 4: Laufwerkstyp :kein Laufwerk

Aktuelle Laufwerksparameter
Laufwerk: Status(HEX)/Laufwerke/Seitenzahl/Sektoren/Spuren
0:0/2/1/18/79
Laufwerk: Status(HEX)/Laufwerke/Seitenzahl/Sektoren/Spuren
1:0/2/1/15/79
Laufwerk: Status(HEX)/Laufwerke/Seitenzahl/Sektoren/Spuren
2:0/2/14/226/231
Laufwerk: Status(HEX)/Laufwerke/Seitenzahl/Sektoren/Spuren
3:0/2/15/243/240
Laufwerk: Status(HEX)/Laufwerke/Seitenzahl/Sektoren/Spuren
4:7/0/0/0/0

Bildschirmmodus : 3 HEX
Bildschirmseite : 0
Zeichen/Zeile   : 80
```

```
/* HC08IB0.C */

/*
 * Testen von BIOS-Interrupts
 * ==========================
 */
#include <dos.h>
#include <bios.h>
#include <stdio.h>
#include <conio.h>
#include types.h

VOID main(VOID)
{
  union REGS r;
  CHAR temp[20];
  INT lfw;

  printf("Testen einiger BIOS-Interrupts\n");
  printf("Speicherkapazität feststellen, "
         "Interrupt 0x12\n");
  int86(0x12,&r,&r);
  printf("Der Speicher ist %u kByte groß\n", r.x.ax);
  printf("Ausstattung feststellen, Interrupt 0x11\n");
  int86(0x11,&r,&r);
  printf
    ("Bitkombination gelesen : %16x HEX\n",r.x.ax);
  printf
    ("Bitkombination gelesen : %16s BIN\n",
    itoa(r.x.ax,temp,2));
  printf
    ("Bitkombination         : ddsgsssDffvvrr7f\n");
  printf
    ("Laufwerk               : %i\n",
    r.x.ax & 0x0001);
  printf
    ("Arithmetik-Prozessor   : %s\n",
    (r.x.ax & 0x0002) ? "nein" : "ja");
  printf
    ("RAM auf Hauptplatine   : %i x16k\n",
    (r.x.ax & 0x000C)2 );
  printf("Bildschirmmodus        : ");
  switch ((r.x.ax&0x0030)4)
  {
  case 0: printf("unklar\n"); break;
  case 1: printf("40x25 Farbe\n"); break;
  case 2: printf("80x25 Farbe\n"); break;
  case 3: printf("80x25 s/w\n"); break;
  }
  printf
    ("Anzahl der Laufwerke   : %i\n", (r.x.ax & 0x00C0)6);
  printf
    ("DMA möglich            : %s\n", (r.x.ax & 0x0100) ? "nein" : "ja");
  printf
    ("Anzahl der RS232-Karten: %i\n", (r.x.ax & 0x0E00)9);
  printf
    ("Game-Port vorhanden    : %s\n", (r.x.ax & 0x1000) ? "nein" : "ja");
  printf
    ("Serieller Drucker      : %s\n", (r.x.ax & 0x2000) ? "nein" : "ja");
  printf
    ("Druckeranzahl          : %i\n\n", (r.x.ax & 0xC000)12);
  printf("Laufwerke feststellen\n");
  for (lfw=0; lfw; lfw++)
  {
    r.h.ah=0x15;
    if (lfw) r.h.dl=lfw;
    else r.h.dl=lfw+0x80-2;
    int86(0x13,&r,&r);
    printf("Laufwerk %i: Laufwerkstyp :",lfw);
    switch(r.h.ah)
    {
    case 0:
      printf("kein Laufwerk\n");
      break;
    case 1:
      printf("Floppy ohne Diskettenwechselsignal\n");
      break;
    case 2:
      printf("Floppy mit Diskettenwechselsignal\n");
      break;
```