Hardwarenahe Programmierung in C/C++

Interrupts und Mausprogrammierung

Christian Zahler

Interrupts

Prinzipiell versteht man unter dem Begriff „Interrupt“ ein Signal, welches die CPU bei der Abarbeitung eines Programms unterbricht.

Man unterscheidet:

- **Hardware-Interrupts:** sind Signale, die von externen Geräten (Tastatur, Bildschirm, Drucker) ausgelöst werden und die CPU bei ihrer laufenden Arbeit unterbrechen. An dieser Stelle wird zunächst in der sogenannten Interrupt-Tabelle (die sich im untersten RAM-Bereich – Adresse 0000h–0400h, das sind 1024 Byte – befindet und auch geändert werden kann) nachgesehen. Dort befindet sich die Adresse des entsprechenden BIOS-Programms (Interrupt Service Routine = ISR), welches dann aufgerufen wird.
- **Software-Interrupts:** von Programmen ausgelöst. Auch diese Interrupts rufen spezielle BIOS-Programme auf. Hardware- und Software-Interrupts arbeiten unabhängig voneinander, dienen aber demselben Zweck – sie regeln die Ein- und Ausgabe von Daten.

Die Interruptsignale müssen über eigene Interrupt-Leitungen übertragen werden. Hier unterscheidet man 16 Leitungen:

IRQ-Leitung	Aufgabe
IRQ 0	Zeitgeber
IRQ 1	Tastatur
IRQ 2	2. Interrupt-Controller 8259A
IRQ 3	COM 2
IRQ 4	COM 1 (meist für Maus oder Modem)
IRQ 5	LPT 2, Soundkarte usw.
IRQ 6	Controller für das Diskettenlaufwerk
IRQ 7	LPT 1 (meist für Drucker)
IRQ 8	Kalenderbaustein
IRQ 9	manche Netzwerkkarten, Grafikkarten usw.
IRQ 10	frei
IRQ 11	frei
IRQ 12	frei
IRQ 13	arithmetischer Coprozessor
IRQ 14	Festplattencontroller
IRQ 15	frei

IRQ = *interrupt request*

Die Kenntnis der Belegung dieser Leitungen ist vor allem dann wichtig, wenn es zu Kollisionen zwischen zwei Peripheriegeräten kommt (etwa Maus und Digitizer), die dieselbe IRQ-Leitung ansprechen.

Es sind 256 verschiedene Interrupts definiert. Jedes Interrupt hat eine eigene hexadezimale Kennummer (INT 01h, INT 02h, ..., INT FFh).

Man unterscheidet:

1. Prozessorinterne Interrupts: INT 00h – 07h

Werden auch als „Traps“ bezeichnet.

INT 00h: wird bei Division durch 0 ausgelöst
INT 01h: Einzelschritt
INT 02h: Nicht maskierbare Unterbrechung (ROM-BIOS)
INT 03h: Unterbrechungspunkt
INT 04h: Zahlenüberlauf
INT 05h: Print Screen

Druckt den Bildschirminhalt aus. Zum Ausdrucken von Grafiken kann man hier eine geeignete Interruptprozedur installieren.

2. Peripherie-Interrupts: INT 08h – 0Fh

INT 08h: Zeitgeber
INT 09h: Tastatur
INT 0Fh: reserviert für Drucker

3. BIOS-Interrupts: INT 10h – 1Ah

INT 10h: Bildschirmsteuerung
INT 11h: Gerätetest
INT 12h: Speichergröße
INT 13h: Disketten-Funktionen
INT 14h: Funktionen für die serielle Schnittstelle
INT 15h: Kassetten-Ein-/Ausgabe (veraltet)
INT 16h: Keyboard

Das BIOS-Interrupt mit der Nummer 16h ist für die Tastatur zuständig. Drückt man eine Taste auf der Tastatur, so entsteht ein elektrischer Impuls, der an eine spezielle Schaltung, die **Tastatursteuerlogik**, weitergeleitet wird. Diese Schaltung erzeugt einen sogenannten Scancode, der genau der gedrückten Taste entspricht. Danach wird ein Interrupt an die CPU gesendet; die CPU löst dann ein BIOS-Programm (eine sogenannte Interrupt-Service-Routine) aus, die dem Tastatursignal den entsprechenden ASCII-Code zuordnet. ASCII-Code und Scancode werden im Tastaturpuffer abgelegt, von wo sie zur Darstellung des Zeichens auf dem Bildschirm oder Drucker verwendet werden können. **Treiberprogramme** wirken genau hier: Sie „stehlen“ das Interrupt; und statt des BIOS-Programms wird das Treiberprogramm ausgeführt. D.h. über Tastaturtreiber kann man eine geänderte Tastaturbelegung erreichen (zB englisch/deutsch). Übrigens: auch **Viren** wirken genauso!

INT 17h: Printer Service

Funktion 0 versucht, ein Zeichen auf den Drucker auszugeben. Im Register AL befindet sich der Zeichencode des auszugebenden Zeichens, in DX die Nummer der angesprochenen Druckers. Hier kann man druckerspezifische Zeichenumsetzungen installieren.

INT 18h: Kassetten-BASIC (veraltet)
INT 19h: Urlader
INT 1Ah: Timer Service

Über Funktion 6 dieses Interrupts kann eine Alarmzeit eingestellt werden (Register CH: Stunden, CL: Minuten, DH: Sekunden). Bei Erreichen dieses Zeitpunktes wird die ISR des Interrupt 4Ah ausgeführt.

4. Anwender-Interrupts: INT 1Bh – 1Fh

INT 1Bh: Aktion nach Drücken von Strg-Pause
INT 1Ch: User Timer
INT 1Dh: Video-Initialisierung
INT 1Eh: Disketten-Parameter
INT 1Fh: Video-Zeichendarstellung (ASCII 128–255)

5. DOS-Interrupts: INT 20h – FFh

INT 20h: Programm beenden, kein Beendigungscode

INT 21h: MSDOS

Einer der wichtigsten Interrupts (*„function request interrupt“*, übersetzt etwa „Anforderung einer Systemfunktion“). Mit diesem Interrupt können ca. 100 Systemfunktionen aufgerufen werden, abhängig davon, welcher Wert beim Aufruf dieses Interrupts im Register AH steht.

INT 22h: Beendigungsadresse des aufgerufenen Programms
INT 23h: Abbruchadresse nach Betätigen der Tastenkombination Strg-Pause
INT 24h: Critical Error Handler

Bei schwerwiegenden Fehlern ruft DOS diesen Interrupt auf. DOS übergibt in den Registern DI und AH Informationen über die Ursache des Fehlers.

Register DI

Fehler-code	Bedeutung
0	Versuch, auf schreibgeschützte Platte zu schreiben
1	Unbekannte Gerätenummer
2	Angesprochenes Gerät nicht bereit
3	Unbekanntes Gerätekommando
4	Datenfehler
5	Falsche Länge in einer Datentruktur
6	Such-Fehler
7	Unbekannte Geräteart
8	Sektor nicht gefunden
9	Kein Papier im Drucker
10	Schreibfehler (falsches Format)
11	Lesefehler
12	Allgemeiner Fehler

Register AH

Hardware-Fehler einer Festplatte werden dadurch angezeigt, daß das Bit 7 des AH-Registers den Wert 0 hat. Die drei niedrigstwertigen Bits dieses Registers zeigen an, ob eine Lese- oder eine Schreiboperation durchgeführt wurde und welche Bereiche der Festplatte betroffen sind:

Fehlercode	Bereich – Zugriffsart
0	Systemdateien, Lesen
1	Systemdateien, Schreiben
2	FAT, Lesen
3	FAT, Schreiben
4	Datei: Verzeichnis, Lesen
5	Datei: Verzeichnis, Schreiben
6	Dateibereich, Lesen
7	Dateibereich, Schreiben

INT 25h: „absolutes Lesen“ von Festplatte: Lesen CX Sektoren nach DS:BX
INT 26h: „absolutes Schreiben“ auf Festplatte: Schreiben CX Sektoren auf DS:BX
INT 27h: Programm beenden, Programm bleibt im Speicher
INT 28h: DOS Timeslice

Dieser Interrupt wird immer dann ausgelöst, wenn DOS gerade nichts zu tun hat. Dieses Interrupt kann nur in speicherresidenten Programmen verwendet werden.

INT 2Fh: Multiplex Interrupt

Installierung speicherresidenter Programme.

INT 33h: Maustreiberaufrufe
INT 3Dh: Vorhandene Datei öffnen
INT 80h – INT FFh: nicht belegt.

Auslösen von Interrupts

Interrupts haben oft verschiedene „Funktionen“, die man durch Belegen der Register auslöst.

Jeder Speicher besteht aus einer Anzahl von Speicherplätzen. Damit Betriebssysteme und Programme aber wissen, „wo es weitergeht“, d.h. die gespeicherten Dinge wieder abgerufen werden können, braucht man Speicheradressen, ähnlich, wie es Adressen gibt, um eben bestimmte Personen erreichen zu können.

Die Größe eines Speicherbereiches, auf den ein Prozessor zugreifen kann, hängt davon ab, wie viele Adreßleitungen er hat. Die Intel-Prozessoren im alten XT 8086/8088 haben 20 Adreßleitungen und können daher

2^{20} Byte = 1 MByte direkt adressieren.

Die Adresse ist also eine 20 Stellen lange Binärzahl oder eine 5 Stellen lange Hexadezimalzahl, z.B. $2FA89_{16}$.

Die Adressen werden in den Registern abgespeichert, das sind sehr wichtige RAM-Bereiche direkt am Prozessor-Chip, die als Zwischenspeicher für Daten und Befehle dienen.

Im 8088 existieren folgende 14 Register:

- vier **Hauptregister**: 16 bit breit, werden mit 16 bit breitem Datenbus versorgt (AX, BX, CX, DX)

AX	Arithmetische und logische Operationen
BX	Adressierung der Datenstrukturen
CX	Schleifenzähler
DX	Ein- und Ausgabeadressen; Erweiterung des AX-Registers

- vier **Segment-Register**: legen die „Speichersegmente“ fest (CS, DS, SS, ES; so befindet sich im Register CS das Code-Segment, in dem die Programmbefehle abgelegt sind, und im Register DS das Daten-Segment, wo sich Daten befinden. -> Es kommt zu keiner Vermischung zwischen Befehlen und Daten)

CS	Code-Segment (Programmanweisungen)
DS	Daten-Segment
SS	Stack-Segment
ES	Extra-Segment (meist für Daten)

- vier **Offset-Register** = „Zeiger“-Register, die die Adressen des Hauptspeichers beinhalten (SP, BP, SI, DI)

SP	Stack Pointer

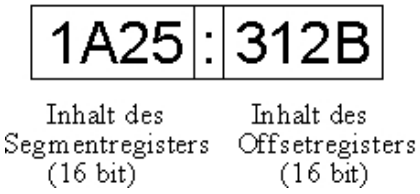
BP	Base Pointer
SI	Source Index (Quelladresse)
DI	Drain Index (Zieladresse)

- zwei **Kontrollregister: Instruction Pointer** (IP; zeigt die Adresse des Befehls an, welcher in einem Programm als nächstes abgearbeitet werden muß) und **Statusregister** (PSW = *processor status word*; enthält eine Reihe von Zahlen = **Flags**, die den Zustand – den Status – des Rechners angeben).

Die "Endungen" der Registerbezeichnungen geben bereits Informationen über deren Verwendung:

- "X" = allgemeine Register
- "P" oder "I" = Adressen
- "S" = Segmentregister (für Segmentadressen)

Die Adressen werden hexadezimal angegeben. Da die Adresse 20 bit lang ist, die Adreßbusbreite im XT aber nur 16 bit beträgt, hilft man sich mit einem „Trick“: Man teilt die Adresse in zwei Teile:



Errechnung der tatsächlichen 20 bit-Adresse: Segmentadresse * 16 + Offsetadresse

```
   1A25.
 + 312B
------
   1D37B
```

Das Multiplizieren der Segmentadresse mit 16 geschieht einfach dadurch, dass man sie (im Hexadezimalsystem) um eine Stelle „nach links verschiebt“.

Das Auslösen eines Interrupts von einem C++-Programm aus geschieht durch den Befehl

```
int86( 0xnn, &register_in, &register_out);
```

0xnn ist die hexadezimale Interrupt-Nummer (für die Maus eben 0x33, für den Drucker 0x17 etc.).

reg ist eine Record-Variable vom Typ REGS; sie enthält als Teilelemente die Registerbezeichnungen in folgender Form:

```
union REGS
{ struct WORDREGS x;
  struct BYTEREGS h;
}
```

wobei

```
struct WORDREGS {
  unsigned int ax, bx, cx, dx;
  unsigned int si, di, cflag, flags;
}
struct BYTEREGS {
  unsigned int al, ah, bl, bh;
  unsigned int cl, ch, dl, dh;
}
```

Definiert man also eine Registervariable reg vom Typ REGS, so können die Register können einzeln gesetzt werden, zum Beispiel

```
reg.x.ax = 3
```

Programmierung der Maus – Interrupt 0x33

Die Programmierung der Maus erfolgt üblicherweise durch Ansprechen der Routinen des Interrupts 0x33. (0x bedeutet, dass die Zahl 33 als hexadezimal anzusehen ist.) Der Interrupt 0x33 enthält mehrere Funktionen; hier die Bedeutung für den Microsoft-Maus-Treiber (andere Maustreiber können andere Wirkungen zeigen!):

ax	Funktion
0	überprüft die ordnungsgemäße Installation der Mausroutinen Rückgabewerte: AX: Status0 ... Maus/Maustreiber nicht installiert 1 ... Maus/Maustreiber installiert BX: Anzahl der Maustasten –1.. zwei Tasten 0 ... mehr oder weniger als zwei Tasten
1	Mauszeiger anzeigen
2	Mauszeiger verstecken (Achtung: mehrfache Aufrufe dieser Funktion benötigen genauso viele Aufrufe der Funktion 1, um den Mauszeiger wieder sichtbar zu machen!)
3	ermittelt die aktuelle Mausposition Rückgabewerte: CX: x-Koordinate (Spalte) DX: y-Koordinate (Zeile) BX: Tastenstatus (0 = keine Taste, 1 = linke, 2 = rechte, 3 = beide Tasten gedrückt)
4	setzt Mauscursor an eine vorgegebene Stelle am Bildschirm CX: x-Koordinate (Spalte) DX: y-Koordinate (Zeile)
5	zeigt an, wie oft eine Maustaste seit dem letzten Aufruf dieser Funktion gedrückt wurde BX = 0 linke Maustaste BX = 1 rechte Maustaste Rückgabewerte: AX = 0 keine Taste gedrückt AX = 1 linke Taste gedrückt AX = 2 rechte Taste gedrückt AX = 3 beide Tasten gedrückt BX = wie oft wurde die ausgewählte Taste seit dem letzten Funktionsaufruf gedrückt CX = x-Koordinate, an der die Taste zuletzt gedrückt wurde DX = y-Koordinate, an der die Taste zuletzt gedrückt wurde
6	zeigt an, wie oft eine Maustaste seit dem letzten Aufruf dieser Funktion losgelassen wurde BX = 0 linke Maustaste BX = 1 rechte Maustaste Rückgabewerte: AX = 0 keine Taste losgelassen AX = 1 linke Taste losgelassen AX = 2 rechte Taste losgelassen AX = 3 beide Tasten losgelassen BX = wie oft wurde die ausgewählte Taste seit dem letzten Funktionsaufruf losgelassen CX = x-Koordinate, an der die Taste zuletzt losgelassen wurde DX = y-Koordinate, an der die Taste zuletzt losgelassen wurde
7	begrenzt den Bewegungsbereich der Maus horizontal CX: minimale x-Koordinate (Spalte) DX: maximale x-Koordinate (Spalte)
8	begrenzt den Bewegungsbereich der Maus vertikal CX: minimale y-Koordinate (Zeile) DX: maximale y-Koordinate (Zeile)
9	definiert den Mauszeiger im Grafikmodus BX: x-Koordinate des „Hot spot“ im Bitmap CX: y-Koordinate des „Hot spot“ im Bitmap ES:DX: pointer zum Bitmap (16 Zahlenwerte, von denen jeder eine Pixelzeile à 16 Bildpunkte definiert; das niedrigstwertige bit ist das rechte)

10	definiert den Mauszeiger im Textmodus BX: wählt Hardware- oder Software-Cursor 0 Software CX = Bildschirmmaske DX = Cursormaske 1 Hardware CX = erste Zeile des Scans DX = letzte Zeile des Scans Beim Software-Cursor wird folgende Verknüpfung durchgeführt: (Bildschirmfarbe an der aktuellen Bildschirmposition AND Bildschirmmaske) XOR Cursormaske
11	liefert die Mausbewegung in „Mickeys“ (1 Mickey = kleinste meßbare Mausbewegung in einer Richtung) CX: Anzahl der Mickeys horizontal seit dem letzten Aufruf dieser Funktion DX: Anzahl der Mickeys vertikal seit dem letzten Aufruf dieser Funktion
15	definiert die Mausbewegung CX: Anzahl der Mickeys pro 8 pixel horizontal DX: Anzahl der Mickeys pro 8 pixel vertikal

Die Funktionsnummern brauchen nur jeweils in das Register AX geschrieben werden, z.B.

```
reg.x.ax = 0x07;
int86(0x33,&register_in);
```

Bei einer professionellen Mausbibliothek sollte auch eine Routine nicht fehlen, die den Textkursor ausschaltet. Dazu verwendet man den Bildschirm-Interrupt 0x10:

Setzt man das Register AH auf 1, dann bedeuten:

CH	Startlinie des Hardware-Cursors im Textmodus
CL	Endlinie des Hardware-Cursors im Textmodus

Setzt man beide Werte z.B. auf 0x20, so bedeutet das eine Cursorgröße von 0 Pixellinien. Folge: Der Textkursor wird unsichtbar.

```
/* Maustreiber-Routine objektorientiert */

#include <stdio.h>
#include <conio.h>
#include <dos.h>

REGS register in, register out;

class Mausklasse{
   public:
     int x,y;
     int taste;
     int sichtbar;
     void reset(void);
     void verstecken(void);
     void zeigen(void);
     void textcursor off(void);
     void mauscursor setzen(unsigned ctyp,
           unsigned smaske, unsigned cmaske);
     int mausx(void);
     int mausy(void);
     int maust(void);
   }; /* Ende Mausklasse */

void Mausklasse::reset ()
{
  register_in.x.ax = 0;
int86( 0x33, &register in, &register out);
if (register_out.x.ax==0)
    {
      puts("Maus nicht installiert - Programmabbruch");
    }
  register_in.x.ax = 10;
  int86( 0x33, &register in, &register out);
}

int Mausklasse::maust ()
{
  register_in.x.ax = 3;
  int86( 0x33, &register_in, &register_out);
  return register out.x.bx;
}

int Mausklasse::mausx ()
{
  register in.x.ax = 3;
  int86( 0x33, &register_in, &register_out);
  return register out.x.cx;
}

int Mausklasse::mausy ()
{
  register in.x.ax = 3;
  int86( 0x33, &register in, &register out);
  return register_out.x.dx;
}

void Mausklasse::mauscursor setzen(unsigned ctyp,
   unsigned smaske, unsigned cmaske)
{
  register_in.x.ax=0x0A;
  register in.x.bx=ctyp;
  register_in.x.cx=smaske;
  register in.x.dx=cmaske;
  int86(0x33, &register_in, &register_out);
}

void Mausklasse::verstecken ()
{
  if (sichtbar)
  {
    register in.x.ax = 2;
    int86( 0x33, &register_in, &register_out);
    sichtbar=!sichtbar;
  }
}

void Mausklasse::zeigen ()
{
  if (!sichtbar)
  {
    register_in.x.ax = 1;
    int86( 0x33, &register in, &register out);
    sichtbar=!sichtbar;
  }
}

void Mausklasse::textcursor off ()
{
  register_in.h.ah = 0x01;
  register_in.h.ch = 0x20;
  register in.h.cl = 0x20;
  int86( 0x10, &register_in, &register_out);
}

void main()
{
Mausklasse maus;  /* maus ist eine INSTANZ von Mausklasse */
textbackground(BLUE);
clrscr();
maus.sichtbar=0;
maus.textcursor off();
maus.mauscursor_setzen(0,0x74ff,0x7f00);
  /* 1. Parameter: 1 = Hardwarecursor (Block- oder Unterstrich)
        0 = Softwarecursor (kann festgelegt werden)
     2. Parameter: Screenmaske
        7  ... Vordergrundfarbe
        4  ... Hintergrundfarbe
        ff ... ASCII-Zeichen 255
     3. Parameter: Cursormaske */
maus.reset();
maus.zeigen();
cprintf("Text");
do {} while (maus.maust()!=1);
   /* Abbruch, wenn linke Maustaste gedrückt */
maus.verstecken();
}
```

Aufgabenstellungen

Erweitern Sie diese Sammlung um folgende Methoden:

1. Einen passenden Konstruktor, der die Startwerte richtig setzt

2. `int Mausklasse::innen (int x1,int y1, int x2, int y2)`
Überprüft, ob sich die Maus innerhalb eines Rechtecks befindet.

3. `void Mausklasse::rechteck (int x1,int y1, int x2, int y2)`
Schränkt die Bewegung der Maus auf einen Rechtecksbereich ein.