RSM

In der c't 1/87 und 2/87 gab es interessante Artikel über resident system module für CP/M 2.2, vergleichbar mit den RSX für CP/M 3. Es handelt sich hier um nachladbare kleine Programme, die etwa BDOS oder BIOS erweitern, wie zusätzliche Gerätetreiber o.ä.

CP/M 2 lernt dazu. Modulare Systemerweiterungen auch für das 'alte' CP/M von Eckhard Lieber und Thomas von Massenbach Teil 1 c't 1987, Heft 1, S. 124-135; Teil 2 c't 1987, Heft 2, S. 78-85

Hier kommt das PRL-Format (http://www.seasip.demon.co.uk/Cpm/prl.html) zum Einsatz.

Beide Autoren und auch die c't haben der Veröffentlichung der Artikel im Internet zugestimmt.

Downloads

- rsm.mac, rsm.com: Das RSM-Tool
- bdosinfo.mac, bdosinfo.rsm: protokolliert BDOS-Aufrufe
- daisy.mac
- edlin.mac
- freeze.mac
- unfreeze.mac
- genrsm.bas
- rsm

🔧 Fix Me!

Teil 1

c't 1987, Heft 1, S. 124-135

Bestimmt haben Sie bei der Arbeit mit CP/M 2 schon häufiger die eine oder andere nützliche Systemfunktion vermißt, besonders, wenn Sie einmal ausführlich mit dem moderneren CP/M Plus 'spielen' durften. Nun ist der einfachste Weg zu mehr Komfort, der Umstieg auf CP/M Plus, nicht immer gangbar - nicht jeder CP/M-2- Rechner ist gleichermaßen CP/M-Plus-geeignet. Man kann aber auch dem guten alten CP/M 2 zusätzliche Fähigkeiten verleihen.

Die frühen Versionen des Betriebssystems CP/M bieten, bedingt durch ihr hohes Alter und die zur Zeit ihrer Entwicklung unabdingbaren Speicherplatz- Sparmaßnahmen, nur die notwendigsten Systemfunktionen. Doch trotz der im Vergleich mit modernen Betriebssystemen spartanischen Ausstattung ist CP/M 2.2 noch immer weit verbreitet - um für andere 'Disk Operating Systems' (DOS) ein ähnlich großes Software-Angebot auf die Beine zu stellen, bedurfte es des guten Namens gewisser Hardware-Hersteller - und schließlich gab es auch bei CP/M Weiterentwicklungen. Eine davon mündete in die letzte CP/M-Version für die 8- Bit-Prozessoren 8080/Z80, die Version 3.0 (CP/M Plus). Daneben wurden in Anwenderkreisen diverse Konzepte für Systemerweiterungen 'ausgekocht', um

den von Haus aus recht mageren Komfort von CP/M 1.4 und 2.x zu verbessern. Allerdings weisen nahezu alle diese Konzepte drei gravierende Mängel auf:

- Die meisten Erweiterungen führen umfangreiche Modifikationen der Systemsoftware durch. Aus diesem Grund lassen sie sich nicht beliebig mit anderen Erweiterungen kombinieren, da sie sich gegenseitig behindern oder das ganze System zum Zusammenbruch bringen können.
- Die Erweiterungen belegen, wie zum Beispiel MicroShell, nicht unerheblich viel Speicherplatz mit Funktionen, die nur selten benötigt werden.
- Eigene Erweiterungen lassen sich nicht einbinden, da keine einheitliche Schnittstellen konvention gegeben ist.

Pflichten

Diese Mängel schränken die Einsatzmöglichkeiten herkömmlicher CP/M- Erweiterungen erheblich ein, so daß ihnen trotz prinzipieller Vorzüge eine größere Verbreitung versagt geblieben ist. Außerdem betreffen die gewünschten Änderungen oft lediglich die elementaren Ein-/Ausgabefunktionen (beispielsweise Zeichenumkodierung für spezielle Drucker) und lassen sich deshalb genauso gut im BIOS einbauen. In vielen Fällen ist das jedoch nicht möglich, zumal BIOS- Änderungen immer mit relativ großem Aufwand und dem Risiko verbunden sind, daß das System hinterher nicht mehr richtig läuft.

Vor diesem Hintergrund reifte der Entschluß, ein neues Konzept für Systemerweiterungen zu realisieren, das im folgenden beschrieben werden soll. Als erstes wurde ein 'Pflichtenheft' erstellt:

1. Die Systemerweiterungen sollen das Kernbetriebssystem (CCP, BDOS und BIOS) unberührt lassen.

2. Sie sollen vom Anwender je nach Bedarf beliebig zusammengestellt werden können.

3. Die Unabhängigkeit von der BIOS-Implementation und der Speichergröße des verwendeten Systems muß gewährleistet sein.

4. Einzelne Funktionen oder Funktionsgruppen sollen in sich abgeschlossene, selbständige Einheiten bilden.

5. Um komplexe Funktionen schnell verfügbar zu machen, soll der Einsatz einer Hochsprache möglich sein.

6. Zur Realisierung der einzelnen Erweiterungen sollen gebräuchliche Assembler und Compiler benutzt werden können.

Grundsätze

Zum besseren Verständnis, wie sich ohne Änderung der Systemteile CCP, BDOS und BIOS Änderungen an den Systemfunktionen vornehmen lassen, sei zunächst dargelegt, wie der Speicher eines Rechners unter CP/M organisiert ist und wie ein RSM in das Betriebssystem integriert wird.

```
    Adresse:    Inhalt:
    ---------------------------------------------------------------------
```

```
    0h:        jp wboot    Sprungvektor ins BIOS, führt Warmstart aus
    3h:        db iobyte    I/O-Byte
    4h:        db cdsk        Disk/User
    5h:        jp bdos        Sprungvektor, über den alle Aufrufe des
                    Betriebssystems erfolgen
    100h:                Beginn der TPA (Speicherbereich für
                    Anwenderprogramme)
    bdos-806h:            Beginn des Standard-CCP (wird von vielen
                    Anwenderprogrammen überschrieben)
    bdos:                Beginn des Kernbetriebssystems (BDOS)
    wboot-3:            Beginn des geräteabhängigen Teils des
                    Betriebssystems (BIOS)
    Die Speicheraufteilung bei CP/M 2. Über die Sprungbefehle auf den
    Adressen 0 und 5 kann man die Lage des Systems im Speicher bestimmen.
```

Das Betriebssystem CP/M liegt grundsätzlich am oberen Ende des verfügbaren Speichers, der bei der Adresse Null beginnen und für CP/M 2.2 mindestens 20 KByte umfassen muß. Transiente Programme dürfen den Speicherbereich von 100h bis maximal zur Adresse bdos-1 nutzen; die Systemfunktionen werden durch einen CALL auf die Adresse bdos aufgerufen. Damit die transienten Programme diese Adresse nicht explizit 'wissen' müssen - was es unmöglich machen würde, ein Programm ohne Änderungen auf Rechnern mit unterschiedlichem Speicherausbau zu verwenden - , erfolgt der Einsprung ins BDOS nicht direkt, sondern über einen Sprungbefehl (Sprungvektor) auf Adresse 5h. Aus diesem läßt sich auch die Größe des nutzbaren Speichers ableiten: Die letzte freie Adresse ist das Sprungziel (16-Bit-Wert auf Adresse 6h) minus eins. So enthalten nur drei Bytes in einem garantiert vorhandenen Speicherbereich alle notwendigen Informationen über die 'großen Unbekannten' BDOS-Einsprung und Speicherausbau.

Wenn man nun das System erweitern will, so muß man hierfür Platz schaffen. Dazu wird zweckmäßigerweise das ohnehin unbestimmte obere Ende des Speichers herangezogen, in dem man die Zieladresse des Sprungs bei 5h um einen noch zu ermittelnden Betrag erniedrigt. Auf diese Weise entsteht unterhalb des CCP ein vor Überschreiben geschützter Speicherbereich, in den beliebige Programme geladen werden können. Es muß lediglich dafür gesorgt sein, daß an der Stelle, auf die der neue Sprungvektor bei Adresse 5h zeigt, ein Weitersprung zum BDOS eingerichtet ist. Alternativ kann man den Sprung abfangen und die BDOS- Unterprogramme durch neuen Code ersetzen - dem Tüftler sind keine Grenzen gesetzt.

Nach demselben Verfahren lassen sich auch mehrere Erweiterungen hintereinanderhängen. Solange das Speicherende nur anhand des Sprungvektors ermittelt wird, ist es völlig gleich gültig, ob dieser noch auf den BDOS-Anfang zeigt oder bereits 'verbogen' wurde.

Um das Entfernen von Erweiterungen braucht man sich normalerweise keine Sorgen zu machen. Da die Warmstartroutine im BIOS unter anderem den Sprungvektor initialisiert, genügt ein Control-C oder JP 0, um die ursprüngliche Speicherkonfiguration wiederherzustellen.

Feinheiten

Dies ist also das Arbeitsprinzip beim RSM-Konzept. Ein paar weitere Punkte lassen sich am besten an einem Beispiel erläutern, einem kleinen Druckertreiber für den TRS80 Daisy Wheel II Printer.

→ **daisy.mac**: Ein erstes Beispiel für ein RSM.

Ähnlich wie eine RSX unter CP/M Plus besitzt ein RSM einen standardisierten Vorspann, der aus zwei Sprungbefehlen, einem Namen und einer Prüfsumme besteht. Der erste der Sprungbefehle führt in das RSM, wo überprüft wird, ob ein BDOS- Aufruf (Funktionsnummer in Register C) abzufangen und umzufunktionieren ist oder nicht. Der zweite Sprung (JP 0) wird beim Laden des RSM initialisiert und stellt den Ausgang zum Betriebssystem oder einem weiteren RSM dar. Auf die bei den Sprungvektoren folgt der Modulname, der acht Zeichen umfaßt, sowie eine 16-Bit- Prüfsumme über den Namen, die der Lader überprüft; nur bei positivem Resultat wird das geladene Modul in das System eingebunden.

Das Beispiel zeigt aber nicht nur die RSM-Formalitäten, sondern auch zwei wichtige programmiertechnische Besonderheiten: Zum einen sollte man, sofern im RSM Unterprogrammaufrufe oder PUSH-/POP-Operationen verwendet werden, stets einen lokalen Stack einrichten (wie es übrigens auch im BDOS geschieht). Bei vielen Anwenderprogram men ist der Stack nämlich so knapp bemessen, daß er bei zusätzlicher Belastung durch das RSM überläuft - in der Folge treten dann irgendwelche obskuren Fehler auf, deren Ursache in der Regel nur sehr schwer zu finden ist.

Zum anderen sollten Aufrufe von BDOS-Funktionen immer über den zweiten Sprung im RSM-Vorspann abgewickelt werden ('call exit') statt über Adresse 5h. Sonst kann es passieren, daß das RSM in sich selbst hängenbleibt. Von dieser Regel gibt es wie üblich auch Ausnahmen, doch dazu ein andermal.

Die zweite Forderung kann bei Verwendung mehrerer RSMs dazu führen, daß beim Laden der RSMs eine bestimmte Reihenfolge einzuhalten ist: Ruft ein RSM eine Funktion aus einem anderen auf, muß dieses in der Kette hinter dem aufrufenden angesiedelt, also vorher geladen worden sein.

Format

Aus den Punkten 2 und 3 des Pflichtenheftes folgt, daß der Speicherbereich für eine Systemerweiterung anders als bei transienten Programmen grundsätzlich nicht feststeht. Das RSM muß daher als verschiebbarer (relokalisierbarer) Objektcode vorliegen, der erst beim Laden in ablauffähigen Maschinencode umgesetzt wird.

Die Wahl fiel auf das von MP/M her bekannte PRL-Dateiformat (Page Relocatable), das auch für die RSX des CP/M Plus zum Einsatz kommt. Anders als REL-Objektcode, wie ihn viele Assembler (RMAC, M80, ...) und Compiler erzeugen, läßt sich PRL- Objektcode nur seitenweise (l Seite = 256 Byte) und nicht byteweise verschieben, da zum Relokali-

sieren nur ein 8-Bit-Offset verwendet wird (für das höherwertige Byte der anzupassenden Adressen). Dafür hat das PRL-Format den gewaltigen Vorteil, daß die Relokalisierungsinformation nicht wie beim REL-Format in den Objektcode eingestreut ist, sondern als Tabelle dahinter steht. Der nichtrelokalisierte PRL-Code sieht also schon wie richtiger Maschinencode aus (ORG-Adresse 100h), was man von REL-Code nicht behaupten kann, und kann demzufolge mit einem gewöhnlichen Debugger ausgetestet werden; man muß ihn nach dem Einlesen lediglich um 100h - die Größe des Vorspanns - nach unten verschieben.

PRL- Dateien lassen sich mit dem LINK80 von Digital Research unmittelbar aus REL-Dateien erzeugen. Falls kein LINK80 zur Verfügung steht, erfüllt das BASIC- Programm GENRSM denselben Zweck, wenn auch etwas umständlicher: Es erwartet als Eingabe zwei relokalisierte Objektcode-Dateien des RSM mit unterschiedlichen Startadressen (100h und 200h).

Dazu ist der RSM-Quelltext zunächst wie für ein transientes Programm mit ORG 100H zu

assemblieren. Anschließend muß man mit dem Editor die Argumente aller ORG-Anweisungen um 100h erhöhen und den Assembler ein zweites Mal bemühen. Sofern der Assembler REL-Dateien erzeugt, müssen die beiden Objektdateien noch gelinkt oder 'geladen' werden.

Nun kann man GENRSM starten, das die Eingabedatei 1 (ORG-Adresse 100h) in die RSM-/PRL-Ausgabedatei überträgt und dabei byteweise mit der Eingabedatei 2 (ORG- Adresse 200h) vergleicht, bei Bedarf das zugehörige Bit in der Relokalisierungstabelle setzt und zum Schluß die Tabelle sowie die Dateilänge in die Ausgabedatei schreibt. Eine Verwechslung der Eingabedateien führt zum Abbruch, ebenso wie der Fall, daß man beim Ändern der ORG-Anweisung nicht auf gepaßt und einen anderen Wert als 200h eingetragen hat.

Dagegen erkennt GENRSM nicht, wenn in den Eingabedateien die Datenbereiche des RSM (Buffer, Stack) nicht vorhanden oder abgeschnitten sind. Manche Assembler beziehungsweise Linker übernehmen mit DBFS (DS) reservierte Bereiche, die am Ende des Quelltextes stehen, nicht in die (COM-)Ausgabedatei, um diese nicht unnötig aufzublähen. Dem kann man jedoch sehr einfach abhelfen, indem man das letzte vom RSM belegte Byte mit DEFB oder DEFW (DB/DW) initialisiert.

Bei Verwendung von Z80-Assemblern und -Linkern, die nur absolute Codesegmente erzeugen können (beispielsweise ZASM und ZLINK), muß im BASIC-Programm an den beiden vermerkten Stellen ('Siehe Text') noch der Offset von 256 dazuaddiert werden. Denn in diesem Fall beginnt der RSM-Code in der Eingabedatei 2, bedingt durch die ORG-200H-Anweisung, erst 256 (100h) Bytes nach dem Dateianfang.

Weiterhin kann die im BASIC-Programm vorgegebene Dateiklasse für die Eingabedateien (BIN), je nach Linker, in 'COM' zu ändern sein.

Um die Vorgehensweise bei den unterschiedlichen Assemblern und Linkern noch einmal zu verdeutlichen, ist oben ein kleines Beispiel abgedruckt, das alle notwendigen Quelltextänderungen und Benutzereingaben zur Erzeugung des RSM 'MODUL' wiedergibt. Dieses RSM lenkt die Druckerausgaben eines Programms (BDOS- Funktion 5) auf den Punch-Kanal um (BDOS-Funktion 4).

→ **rsm**: Drei Beispiele, wie man mit verschiedenen Assemblern und Linkern eine RSM-Datei im PRL-Format erzeugt.

→ **genrsm.bas**: Dieses MBASIC-Programm erzeugt PRL-Dateien, wenn der LINK80 nicht zur Verfügung steht.

```
    Record 1, Byte 0     : 0
    Record 1, Byte 1,2     : Länge des Programms in Bytes
    Record 1, Byte 3     : 0 (nicht benutzt)
    Record 1, Byte 4,5     : Zusätzlich benötigter Speicher
    Record 1, Byte 6 - 127    : 0
    Record 2, Byte 0 - 127    :0
    Record 3 und folgende enthalten den eigentlichen Maschinencode.
        Direkt anschließend folgt eine Bit-Tabelle, in der alle
adreßabhängigen
        Bytes mit jeweils einem Bit vermerkt sind, beginnend mit dem
        höchstwertigen Bit des ersten Bytes der Tabelle. Ein gesetztes Bit
        bedeutet, daß das zu gehörige Byte mit einem Offset zu versehen ist.
Der
        Offset ist das höherwertige Byte der Zieladresse - 100h.
```

So ist das PRL-Dateiformat aufgebaut. Dem verschiebbaren Maschinencode ist ein 256-Byte-Header vorangestellt.

Laden

Und wie kommen die im PRL-Format vorliegenden RSMs nun an ihren Platz? Da der CCP bei den CP/M-Versionen vor 3.0 auf Systemerweiterungen noch nicht eingerichtet ist, mußte ein spezielles Ladeprogramm RSM.COM geschrieben werden. Er erfüllt folgende Aufgaben (in der Reihenfolge der Ausführung):

CPU, CP/M und Datei testen

Bei einer 8080/85-CPU, einer Nicht-PRL-Datei (erstes Byte ungleich 0) oder unter CP/M Plus verweigert der Lader die Mitarbeit.

Platz reservieren

Die Summe der Angaben in den Bytes 2, 3 und 4, 5 der PRL-Datei ergibt die Länge des RSM. Um diesen Wert wird das Ende der TPA vorverlegt, es sei denn, die verbleibende TPA ist kleiner als 8 KByte (Abbruch). Beim ersten zu ladenden RSM ist noch die Länge des CCP zu berücksichtigen (2 KByte beim Standard-CCP), der erhalten bleiben muß.

RSM laden und relokalisieren

Der Lader liest die in der PRL-Datei angegebene Anzahl Programmbytes unmittelbar in den Zieladreßbereich ein und relokalisiert sie an Ort und Stelle. Ein vorzeitiges Ende der PRL-Datei führt zum Abbruch.

Prüfsumme abfragen

Anschließend wird die Prüfsumme über den Modulnamen gebildet und mit der im RSM- internen verglichen. Bei Ungleichheit war es vielleicht doch keine RSM-PRL- Datei… (Abbruch).

Sprungvektoren einrichten

Die vorletzte Aktion des Laders (nach der 'O.K.'-Meldung) ist das Setzen des neuen Sprungvektors auf Adresse 5h und des 'Fortsetzungs'vektors im RSM. Ab diesem Zeitpunkt ist das Modul in das System eingebunden und geschützt.

Modul aktivieren

Manche RSMs benötigen eine Initialisierung, die zweckmäßigerweise durch den ersten Auf ruf (erster 'BDOS-Call' nach dem Einbinden ins System) ausgelöst werden sollte. Dazu fordert der Lader zum Schluß einen 'Direct Console Input' an (BDOS-Funktion 6 mit E = FFh), dessen Ergebnis ignoriert wird.

Die Forderung, den CCP zu erhalten, ergibt sich vor allem daraus, daß CP/M zwei Wege zur Verfügung stellt, aus einem Anwenderprogramm zur Kommandoebene des Systems zurückzukehren - den Warmstart (JP 0) und den Rücksprung (RET) zum CCP. Beim RSM-Lader ist sogar nur der zweite Weg gangbar, da ein Warmstart unter anderem den Sprungvektor zum BDOS auf die ursprüngliche Adresse zurückstellen und damit ein gerade eingebundenes RSM sofort wieder 'entbinden' würde. Außerdem: Was nützt einem ein System mit RSM(s), wenn man wegen des fehlenden CCP kein Programm mehr aufrufen kann?

Um den Lader zu starten, ist eine Zeile nach dem Muster

A>RSM <filename>

einzugeben. Als Dateiklasse wird 'RSM' angenommen, so fern sie nicht explizit angegeben ist. Fehlt der Dateiname oder enthält er Jokerzeichen ('*'/'?'), bricht RSM.COM ab. Außer bei falscher CPU endet der Programmlauf immer mit einer Aufzählung der eingebundenen RSMs und ihrer Anfangsadressen.

Es friert

Wie gesagt bleibt RSMs ohne besondere Vorkehrungen nur bis zum nächsten Warmstart aktiv. Mitunter besteht jedoch der Wunsch oder auch die Notwendigkeit, die Erweiterung(en) zu erhalten, beispielsweise, wenn darüber eine submit- ähnliche Ablaufsteuerung per Eingabedatei stattfindet (dem GET-Befehl von CP/M Plus vergleichbar).

Diese Aufgabe übernimmt das RSM 'FREEZE'. Dazu muß FREEZE, und das ist leider nicht zu umgehen, die 'erste Pflicht' verletzen und in das existierende System eingreifen: Es klinkt sich in die BIOS-Sprungleiste ein, indem es den Sprung zur Warmstart-Routine bei Adresse bios + 3 auf eine eigene Routine umlenkt. Diese errichtet einen lokalen Stack, initialisiert den BDOS-Sprungvektor auf den Anfang von FREEZE, setzt das Diskettensystem zurück (BDOS-Funk tion 13) und verzweigt zum zweiten Einsprang in den CCP (Adresse ccp + 3, CCP-Aufruf mit Löschen des Eingabepuffers).

→ **freeze.mac**: Damit man RSMs nicht nach jedem Warmstart neu laden muß, können sie mit diesem Spezial-RSM 'eingefroren' werden.

Nach dem Aktivieren von FREEZE sind alle vorher geladenen RSMs warmstartsicher installiert, ebenso der CCP und das BDOS, die somit beim 'Warmboot' nicht mehr nachge laden zu werden brauchen - ein recht erfreulicher Randeffekt. Selbstverständlich können hinterher weitere Moduln geladen werden, die allerdings nicht mehr geschützt sind, es sei denn, man lädt FREEZE erneut. In dem Fall bleibt jedoch die alte Kopie von FREEZE im Speicher und belegt eine wertvolle Page.

Will man das ursprüngliche System wiederherstellen, kann man die 'Brutalmethode' anwenden (Kaltstart des Systems) oder etwas subtiler vorgehen, indem man FREEZE 'in den Selbstmord treibt': Bei Aufruf der BDOS-Funktion 255 restauriert FREEZE nämlich den alten Warmstartvektor in der BIOS-Sprungleiste und löst einen Warmstart aus. Zuvor wird noch ein BDOS-Call Nummer 254 abgesetzt, der die bis dato geschützten RSMs über den bevorstehenden 'Rausschmiss' informiert. Diese können dann eventuell notwendige Sicherungsmaßnahmen durchführen.

Mit dem anschließenden Warmstart steht die TPA wieder in alter Größe zur Verfügung. 'Alt' heißt hier: vor dem Laden des gerade deaktivierten FREEZE. Befinden sich mehrere FREEZE-Moduln im Speicher, wird nur der Bereich bis zum nächsten FREEZE freigegeben. Man kann also gezielt mehrere Blöcke von Erweiterungen 'einfrieren' und einzeln wieder 'loseisen' - eine Möglichkeit, die unter Umständen wertvoller sein kann als der vorhin zitierte wertvolle Speicherplatz.

Der Aufruf der BDOS-'Anti- FREEZE'-Funktion kann von jedem Programm aus erfolgen (anstelle des JP 0). Selbst wenn kein FREEZE-Modul im Speicher steht, sind Probleme wegen der Funktionsnummer nicht zu befürchten, weil das BDOS bei unbekannten Funktionen so fort 'returnt'. Alternativ ist das folgende Minimalprogramm geeignet.

Fortzusetzen

Mit den hier vorgestellten Programmen ist das Handwerkszeug zum Erstellen und Arbeiten mit RSMs komplett, soweit es die Programmierung in As semblersprache betrifft. Auf weitere Aspekte des RSM-Konzeptes - etwa die Realisierung komplexerer Funktionen in einer Hochsprache (Fortran) - können wir aus Platzgründen erst im zweiten Teil des Artikels eingehen. Dort werden auch noch ein paar nützliche Applikationen zu finden sein, zum Beispiel ein Zeileneditor, der die BDOS-Funktion 10 (Read Console Buffer) ersetzt und das Zurückholen und Editieren der letzten Eingaben erlaubt.

Literatur

[l] Bernd Pol, Vom Umgang mit CP/M, Eine allgemeinver ständliche Einführung, IWT Verlag, Vaterstetten, 1983

[2] Allan R.Miller, Mastering CP/M, Sybex Inc., Berkeley, California, 1983

[3] Rodnay Zaks, How to Pro gram the Z80, Sybex Inc., Ber keley, California, 1980

[4] LINK80 Operator's Guide, Digital Research, 1980

[5] MP/M II Operating System, Programmer's Guide, Digital Research, 1981

[6] CP/M Operating System, Ma nual, Digital Research, 1982

[7] Hans-Peter Sauer, Veränderlich, Systemerweiterungen unter CP/M-Plus, c't 4/86

→ **rsm.mac**: Das Ladeprogramm für die RSMs ist in zwei Versionen abgedruckt: Als Assembler-Quelltext für diejenigen, die alles genau wissen wollen, …
… und als Hexdump für diejenigen, die es schnell und ohne viel Aufwand zum Laufen bringen wollen.

Teil 2

c't 1987, Heft 1, S. 78-85

Nachdem Sie vielleicht schon die ersten Erfahrungen mit RSMs und ihrer Programmierung in Assembler gesammelt haben, wollen wir nun wie versprochen noch ein paar 'Feinheiten' nachreichen. Zum einen ist damit die Möglichkeit gemeint, für die Erstellung eines RSM auch Hochsprachen-Compiler einzusetzen; zum anderen, was besondere Einsatzfälle an besonderen Maßnahmen verlangen. Und schließlich drucken wir zwei 'Leckerbissen' für CP/M-User und -Programmierer ab - die RSMs EDLIN und BDOSINFO.

Will man komplexe Funktionen als RSM implementieren, so ist es sinnvoll, komplizierte Routinen, die in Assembler nur sehr schwierig realisierbar sind, in einer höheren Programmiersprache zu erstellen. Voraussetzung dafür ist, daß der Compiler relokalisierbaren Objektcode erzeugt, der mit dem in Assembler geschriebenen RSM-Rumpf zusammen'gelinkt' und mit einem der vorgestellten Verfahren in eine PRL-Datei umgesetzt werden kann. Geeignet ist zum Beispiel der Fortran-Compiler F80 von Microsoft, der darüber hinaus über gut dokumentierte Schnittstellen zu Assembler-Programmen verfügt.

Typische Anwendungen für Hochsprachen-Unterprogramme sind Berechnungen (floating point) oder

komplexe Ein-/Ausgaben. Als triviales Beispiel möge ein Fortran- Programm dienen, welches einen Integer-Wert formatiert auf den Bildschirm ausgibt. Angenommen, der Quelltext des aufrufenden Assembler-Programms liege als AS.MAC und der des Fortran-Unterprogramms als FORTRAN.FOR auf der Diskette vor, dann erzeugen M80, F80 und LINK80 mit folgenden Eingaben das RSM (MODUL.RSM):

A>F80 =FORTRAN A>M80 =AS A>LINK MODUL.RSM = AS,FORTRAN,FORLIB[S][OP]

```
Fortran-Unterprogramme von einem Assembler programm aus aufzurufen, ist
mit dem F80-Compiler gar nicht so kompliziert, wie dieses Beispiel einer
formatierten INTEGER-Ausgabe zeigt.
```

Mehr BIOS

BIOS-Änderungen per RSM stehen eigentlich im Widerspruch zu Punkt 1 des Pflichtenheftes, wonach die Erweiterungen das bestehende System unberührt lassen sollen. Sie stellen aber ein besonders weites Betätigungsfeld dar, weil auf diese Weise allen CP/M-Benutzern geholfen ist, deren BIOS sich in einem ROM befindet oder sich wegen mangelnder Dokumentation nicht ändern läßt. (Besser ist es natürlich, wenn das BIOS als Source in einer Datei vorliegt, aber die Politik mancher Hersteller steht dem bekanntlich entgegen...)

Die Arbeitsweise von RSMs mit BIOS-Erweiterungen entspricht der des FREEZE- Moduls aus dem letzten Heft: Der vom Lader ausgeführte BDOS-Call #6 wird abgefangen, der zur gewünschten BIOS-Funktion gehörende Vektor aus der BIOSSprungleiste gelesen und gespeichert. Anschließend ist der Vektor so zu 'verbiegen', daß er auf die neue Funktionsroutine im RSM zeigt. Erkennt das RSM den (nur) von FREEZE initiierten BDOS-Call #254, restauriert es den alten Vektor, gibt dann aber im Unterschied zu FREEZE den BDOS-Call weiter, damit sich auch die folgenden RSMs auf ihre 'Entladung' vorbereiten können.

Daraus folgt für den Umgang mit solchen Erweiterungen zweierlei: Zum einen verlangen BIOS-ändernde RSMs unbedingt ein nachfolgend geladenes FREEZE-Modul. Sonst ist ein Systemabsturz vorprogrammiert, da die RSMs den nächsten Warmstart nicht überstehen, die Änderungen in der BIOS-Sprungleiste aber schon. (Alternativ kann man die Freeze-Funktion auch mit in das betreffende RSM einbauen.) Zum anderen spielt es keine Rolle, ob noch weitere RSMs im Speicher stehen, die die gleiche BIOS-Funktion modifizieren. Das Modul, das als letztes den BIOS-Sprungvektor ändert, ist das erste, das die Änderung wieder rückgängig macht. Die alte Routine 'wiederzufinden' ist auch bei beliebiger Verschachtelung möglich.

Eingabekomfort

Kommen wir nun zu den beiden RSM-Anwendungen. Die erste, EDLIN, ist ein komfortabler Zeileneditor, der die BDOS-Funktion 'Read Console Buffer' (Nummer 10) ersetzt. Nach dem Laden dieses RSM werden alle Programme, die die BDOS- Funktion 10 aufrufen, von EDLIN statt vom BDOS bedient, so auch der CCP.

EDLIN besitzt einen zirkulär organisierten Puffer für die letzten sechs Eingabezeilen, die mit den WordStar-kompatiblen Kommandos Control-E (letzte Eingabezeile) und Control-X (nächste Eingabezeile) in die aktuelle Eingabezeile übernommen werden können. Ein Edieren innerhalb der Eingabezeile, meistens zur Korrektur fehlerhafter Eingaben, ist mit ebenfalls WordStar-kompatiblen

Befehlen möglich; weiterhin verfügt EDLIN über einen Insert-Modus, dessen Aktivierung auf dem Bildschirm sichtbar gemacht werden kann (etwa durch Umschalten der Cursor- Darstellung, siehe Hinweise im Listing). Die Tabelle 'EDLIN-Kommandos' gibt einen Überblick über die implementierten Control-Codes und ihre Wirkung.

EDLIN-Kommandos	
Ctrl-V	Insert on/off
Ctrl-S, Ctrl-H	Cursor links
Ctrl-D	Cursor rechts
Ctrl-A	Cursor zum Zeilenanfang
Ctrl-F	Cursor zum Zeilenende
Ctrl-E	letzte Eingabezeile aus dem Puffer holen
Ctrl-X	nächste Eingabezeile aus dem Puffer holen
Ctrl-Y	Zeile löschen
Ctrl-G	Zeichen auf Cursor-Position löschen
DEL	Zeichen links vom Cursor löschen
Ctrl-C	Warmstart, falls Cursor in Spalte 1
Ctrl-P	Printer on/off
Ctrl-M (Return)	Eingabe beenden

Das abgedruckte Programm er setzt übrigens nicht nur die BDOS-Funktion 10, sondern auch die Funktionen l (Console Input), 2 (Console Output) und 1 1 (Get Console Status). Diese werden jedoch nicht wie die Funktion 'Read Console Buffer' erweitert, sondern eher im Gegenteil: Anders als die Original-Funktionen werten die Ersatz-Routinen weder Control-C noch Control-S aus, auch die Expansion von TAB-Codes bei der Ausgabe unterbleibt. Im allgemeinen stören diese Ein schränkungen nicht; falls doch, kann man durch Löschen oder 'Kommentarisieren' (';' in die erste Spalte der Programmzeile) der zugehörigen Abfragen zu den Original-Funktionen zu rückkehren. (Betroffen sind die ersten drei CP/JP-Sequenzen nach dem Label 'exec'.)

Das BDOS informiert

BDOSINFO ist ein Hilfsprogramm, das alle BDOS-Aufrufe mit Ausnahme der Zeichen- I/O-Funktionen (Console-, List-, Reader- und Punch- Funktionen) auf dem Bildschirm und eventuell auf dem Drucker protokolliert. Damit läßt sich sehr schön die Arbeitsweise von PIP und anderen Programmen beobachten. Weiterhin ist BDOSINFO sehr nützlich bei der Fehlersuche in nichtlauffähigen Programmen und bei der Eigenentwicklung von CP/M-Programmen.

Programmtechnisch enthält BDOSINFO noch einen kleinen Gag: Es stellt selbsttätig fest, ob sich das aufrufende Programm oberhalb oder unterhalb des eigenen Adreßbereichs befindet (Subtraktion RSM-Anfangs- minus Aufrufadresse), und kann somit unterscheiden zwischen BDOS-Calls von Systembestandteilen (CCP oder höherliegende RSMs) und Anwenderprogrammen (oder tiefer liegenden RSMs). Nur letztere werden 'mitgeschrieben'.

Viel zu tun

Die Möglichkeiten, das CP/M-2-System durch RSMs 'aufzubohren', sind fast nur durch die eigene Phantasie begrenzt. Die Beispiele zu diesem Artikel stellen nur eine kleine Auswahl der denk- und

machbaren Erweiterungen dar. Selbst komplexere Gebilde wie Grafiktreiber oder Mathematik-Pakete, deren Funktionen mit nicht definierten Funktionscodes aufgerufen werden (üblicherweise größer 80h), lassen sich realisieren (das CP/M-Plus-Grafikpaket GSX arbeitet ähnlich).

Einschränkungen ergeben sich lediglich durch das zwangsläufige Schrumpfen der TPA. Verschiedene CP/M-Programme benötigen nun einmal soundso viele freie Kilobytes (zum Beispiel braucht WordStar 3.0 mindestens 40 KByte TPA), die erhalten bleiben müssen. Um Ihnen eine Vorstellung von den Platzverhältnissen unter CP/M 2.2 zu geben: Die Nenn-Systemgröße beinhaltet 2 KByte für den CCP, 3,5 KByte für das BDOS und 1,5 KByte für das BIOS, bei einem 64K-CP/M verbleiben also 57 KByte für die TPA.

Allerdings hat man bei RSMs (wie bei allen Programmen, die kaum Texte enthalten) schon einiges zu programmieren, um in die Größenordnung von Kilobytes zu kommen. Somit braucht man sich normalerweise nur dann Sorgen um den Speicherplatz zu machen, wenn man mehrere größere RSMs gleichzeitig im Speicher halten muß - und diese Fälle sind ziemlich selten. Ob die Sorgen dann auch begründet sind, zeigt am besten ein Versuch.

Wie bereits im ersten Teil des Artikels angedeutet, kann es in Ausnahmefällen erforderlich sein, von der Regel abzuweichen, daß BDOS-Aufrufe in einem RSM nur über den zweiten Sprungbefehl im RSM-Kopf zuerfolgen haben. Diese Ausnahmen zeichnen sich dadurch aus, daß a) mehrere RSMs benötigt werden, b) die gewünschte Funktion nicht in dem aufrufenden RSM enthalten ist (sonst wäre der Umweg über Adresse 5 unnötig) und c) die Reihenfolge der RSMs nicht variabel gehalten werden kann (wenn etwa das aufrufende RSM 'einzufrieren' ist und das andere nicht). Ein Beispiel wäre, daß man die Funktion einer BIOS-Erweiterung, die zusammen mit FREEZE als erstes geladen wird, von dem Vorhandensein bestimmter weiterer RSMs abhängig machen will. Praktisch kommen solche Fälle jedoch kaum vor, hier sind sie nur der Vollständigkeit halber aufgeführt.

Wir danken Herrn Rainer Wagner für die freundliche Unterstützung unserer Arbeit.

→ **edlin.mac**: Etwas länger, aber weit leistungsfähiger als die Original-Read-Buffer-BDOS-Routine: der Zeileneditor EDLIN.

→ **bdosinfo.mac**: Außer zu den sogenannten Charakter-IO-Funktionen hat BDOSINFO zu jeder von einem Anwenderprogramm aufgerufenen BDOS-Funktion etwas zu sagen„.

From:
https://hc-ddr.hucki.net/wiki/ - **Homecomputer DDR**

Permanent link:
https://hc-ddr.hucki.net/wiki/doku.php/cpm/rsm/ct?rev=1319038404

Last update: **2011/10/19 15:33**