CP/M-86およびコンカレントCP/M-86へのexec機能の組み込み

井田 昌之

CP/M-86とコンカレントCP/M-86にexec機能を組み込む方法を紹介している。exec機能とはプログラム中から，他のプログラムを実行する機能である。たとえばLispインタプリタを実行中にエディタを呼び出したりできる。この機能はMS-DOSにはあるが，CP/M-86にはない。(本誌)

プログラム内でのコマンド実行：exec機能

プログラムの内部でコマンド・インタプリタを呼び出し，(キーボードからではなく) 内部からコマンドとして文字列を与え任意に実行できる機能を，C言語のシステム・コール命名法にちなんでexec機能と呼ぶことにする。exec機能があれば組み込みコマンドや，第三者の作成したソフトウェアをそのまま自然に自分のプログラムの一部として実行できる。伝統的な意味で，直接に最も利用価値があるものの一つとしてプリプロセサの形で言語処理系を作ることがある。たとえばCのコードを中間ファイルに出力しておいて，続けて内部的にCコンパイラを呼び出すといった細工が可能になる。

16ビット・パソコンでよく用いられているOSとしてMS-DOSとCP/M-86があげられるが，この両者をexec機能の点で見ると完全に優劣がついている。すなわち，MS-DOSにはexec機能があるがCP/M-86にはない。これはささいなことだが，CP/M-86の使い勝手を悪くしている。ここで述べるexec機能はそうした点を自前で補うことを目的としている。

図1に筆者のシステム上での会話例を示す。Lispを実行途中にCP/Mのコマンドを実行したくなったときには，即座に実行させ，またLispの処理を継続できる。この機能を組み込んだので，Lispの中に動的にたくわえていったリスト構造をメモリにおいたままちょっと寄り道をしたり，あるいは画面エディタを呼び出したりできるようになった。この機能を作り，使い始めてから1年ほど経過し，その間順調に使ってきた。またコンカレントCP/M-86へテスト移植し，そこでも正常に動いているので，それらについて紹介する。

ここに述べる方法はLisp処理系にだけ有効というわけではない。広く一

```
B>LISP

     LISP05   ver.2.8
(C) 1984 Masayuki IDA, Aoyama Gakuin Univ.
CS1CS2DSES = 50004C002C003C00
                CS2(USRCSEG) = 16k for V2

eval> (cpm "dir *.mod")

B: LSUBR1   MOD : LET      MOD : FUNCALL  MOD : LISP     MOD
B: DO       MOD : PRINT    MOD : LSUBR2   MOD : EVAL     MOD
B: OBLIST   MOD : MICS     MOD : LLXIO    MOD : GETCELL  MOD
B: PROG     MOD : READ     MOD : APPLY    MOD
B>^C
Value is...NIL
eval> (cpm "stat lisp.mod")


 Drive B:                          User :  0
 Recs  Bytes FCBs Attributes       Name
   39     6k    1 Dir RW         B:LISP    .MOD
-----------------------------------------------
Total:     6k    1
B: RW, Free Space:          36k
Value is...NIL
```

図1　CP/M-86上で動作しているプログラムから他のプログラムを呼び出した例

筆者は青山学院大学勤務。理工学部経営工学科情報科学研究室に所属している。

般のプログラムに組み込んで用いることができる。

またCP/M-86への組み込みは，CP/M-86の開発者であるDigital Researchおよびデジタルリサーチジャパンとは無関係であり，筆者が試作研究の一部として行ったものでだれが利用してもよいが，筆者も含めてだれもその結果について保証するものではないことをお断りしておく。

まず基本的exec機能を持っているコンカレントCP/M-86を説明し，次にCP/M-86へ移ることにする。

コンカレントCP/M-86のexec機能

コンカレントCP/M-86は，コマンド・インタプリタ呼び出しのためのエントリを用意している。したがってそれを呼び出せばよい。そのための基本命令列は

```
mov   cl, 96 h
mov   dx, offset  command
int   224
```

である。

clレジスタにはコマンド・インタプリタ呼び出しであることを示す96_Hを置き，dxレジスタには実行したいコマンド文字列領域の先頭番地を与える。その領域は，先頭1バイトを0_Hとし，直後にコマンド文字列を続け，最後にまた0_Hを置く。先頭の1バイトにゼロを置く点が常識的な文字列表現と異なると思えばよい。したがって次のようになる。

```
command db  0, 'dir  *. *', 0
```

このexec機能はサブタスクのattachであり，自分自身と並行して先へ進むことができる。要求した実行の結果を待って先へ進みたいとしたら，同期を要求して待たなければならない。そのための一つの方法として，仮想コンソール要求がある。コマンド・インタプリタを起動するとそれがコンソール使用権を握るので，呼び出したプログラムではまったくコンソールI/Oができない状態となる。そこでコンソール要求を出し，これによってタスク終了待ちにできる。このための手順は次のようになる。

```
mov   cl, 92 h
int   224
```

タスク優先度の処理：コンカレントCP/M-86で必要な工夫

これにより，領域commandにセットしておいたコマンドが実行されるが，コンカレントCP/M-86がマルチタスク／マルチウインドウOSであることから生ずる約束があり，それを守らなければ正しく機能しない。それはタスクの優先度である。

コンカレントCP/M-86下で動作するプログラムはOSがタスクとして管理する。もちろんTmpと呼ばれるコマンド・インタプリタも同様である。

優先度は1バイトで表され，通常のユーザ・プログラムの優先度は$0C8_H$である。値が小さい方が優先度が高い。Tmpの優先度は$0C6_H$である。したがって通常は，

優先度（Tmp）＞優先度（ユーザ・プログラム）

となっている。

そこで上述したように，コマンド文字列を与えてコマンドを実行させると，Tmpの優先度の方が高いので，呼び出されたプログラム（コマンド）の実行後，呼び出した側のプログラムに制御が戻らない。コマンド・インタプリタが制御を握ったままになってしまう。

コマンド実行を起動する前に自身の優先度をTmpより高く変更し，その後コマンド実行を行う。制御が帰ってきたら，元の優先度に戻せばよい。

自分自身の優先度を変更するには，clレジスタに91_H，dlレジスタに優先度を入れてシステムを呼び出す。たとえば優先度を$0C4_H$にしたければ次のようにする。

```
mov   cl, 91 h
mov   dl, 0C4 h
int   224
```

これで，優先度をTmpよりも高くできる。

これらをまとめた手順を**図2**に示す。

CP/M-86へのexec機能組み込み

BDOS #47の利用

本来CP/M-86にはexec機能がない。そこで，この機能をOSの標準機能に部分的にパッチすることにより代用させる。このために，BDOS #47（プログラム・チェイン）に白羽の矢をたてた。このプログラム・チェイン機能は，コマンド行となる文字列を与え，そこへ制御を移す機能である。

BDOS #47はプログラムをチェインするためのエントリであり，いくつかのフェイズに分けて仕事をする場合を想定して用意されている。その基本的な手順は次の通りである。

(1) DMAセグメントの設定およびDMAアドレスの設定により，実行すべきコマンドを表す文字列の先頭番地を指定する。

(2) BDOS #47を実行する。

この手順により，BDOS #47を発行したプログラムは消滅し，指定したコマンドを探し，存在すれば実行する。

制御が移行してしまうので，いったんBDOS #47を実行すると呼び出したプログラムがそのときまでにメモリ上に構築した環境・状態は一瞬のうちに消滅してしまう。

BDOS #47の機能の修正

そこでBDOS #47のエントリの中をデバガ（DDT 86）を使って，

(1)制御を取り戻す方法はないか。

(2)メモリ・イメージをそのまま保つにはどうしたらよいか。

の2点を念頭におきながら調べていった。その結果，

①メモリを解放するか否かのスイッチとパラメータがあり，それを調整すれば，呼び出し側プログラムをメモリにおいたままチェインできる。

②チェインをしたあとは，コマンド・インタプリタへまっすぐ入り，処理が進む。warm bootの所（'A>'などのコマンド・プロンプトを出力する所）が事実上の終了点である。そこでコマンド・プロンプトを出力する代わりに，制御を保存しておいた呼び出し側のプログラムに返すとよい。

という2点を推理し，パッチにとりかかった。

作成してみると一応正しく動作した。しかし，②の点については，組み込みコマンドを実行した場合，筆者の方法では，パッチ個所を通らず，'A>'のプロンプトを出してしまう。そこでは更に通常のコマンド実行ができる。呼び出したプログラムは生きているので，コントロールCをタイプすれば呼び出し側プログラムに制御を返すことができる。

```
; 1)  commandに文字列として
;      コマンドをセットする
; 2)  優先度を高くする
            MOV   CL, 91h
            MOV   DL, 0C4h
            INT   224
; 3)  コマンド・インタプリタの呼び出し
            MOV   CL, 96h
            MOV   DX, command
            INT   224
; 4)  コンソール要求による終了待ち
            MOV   CL, 92h
            INT   224
; 5)  優先度を元に戻す
            MOV   CL, 91h
            MOV   DL, 0C8h
            INT   224
            :
command     DB    0, 'cc1 temp.c', 0
            :
```

図2　コンカレントCP/M-86におけるexec機能

すべての場合に，指定したコマンドを一つだけ実行し，終了後即座に呼び出し側プログラムに制御が戻るようにするには，ここで述べる方法をさらに拡張しなければならないが，実際に1年間，この形で使っていて都合がいい場合もあるのでこのままにしている。

実際のプログラム

以上で述べた手順を**図3**に示す。また実際のプログラムを**図4**に示す。

図4の中で，ESレジスタに0040_Hをセットしているが，これは日本電気N 5200-05用CP/M-86のCPM.SYSのセグメント・ベースである。PC 9800

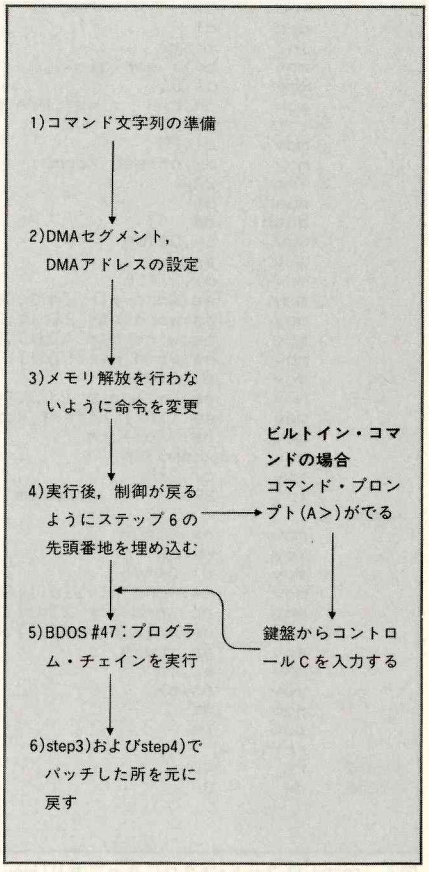

図3　CP/M-86のexec機能の動作概要

用CP/Mの場合には，0040_Hではなく0060_Hとしなければならない。

自分が用いているCP/M-86のCPM.SYSのセグメント・ベースがわからない場合，これを確認するにはいくつかの方法がある。簡単な方法としては，DDT86デバガを呼び出し，D 0：0コマンドで表示される1番先頭の部分を見てみればよい。そこには割り込みのための分岐表がある。2バイトごとに入っている値が，およそCPM.SYSのベースであると思ってよい。

```
;
; ========== CPM ========== subr 1
;                    CP/M-86 command execution ONLY for CP/M-86 V1.1a
        public  xcpm
xcpm:
        shl     bx,1    ! jnc xcpm1   ! retnil
xcpm1:  shl     bx,1    ; get H-address
        mov     ax,cs
        mov     ds,ax
        mov     si,4[bx]        ; get string addr
        mov     cl,1[bx]        ; get string length
xcpmx:  ; --------- another entry ----------
        mov     di,offset xcpm2
xcpm3:
        mov     al,0[si]
        cmp     al,61h
        jc      xcpm5
        xor     al,20h
xcpm5:
        mov     0[di],al
        inc     si      ! inc di
        dec     cl
        jnz     xcpm3
        mov     byte ptr 0[di],0        ; set trailing null
        mov     cl,51
        mov     dx,cs   ; set DMA segment
        int     224
        mov     cl,26
        mov     dx,offset xcpm2
        int     224
        push    es
        push    ds
        mov     ax,0040h
        mov     es,ax
        mov     di,21f5h
        mov     es:word ptr [di],06c6h
        mov     es:word ptr 2[di],2498h
        mov     es:word ptr 4[di],0e900h
        mov     es:word ptr 6[di],0ed54h        ; mov byte[2498],jmp 0f51
        mov     di,0f4ch
        mov     es:byte ptr [di],0eah   ; direct jmpf
        mov     es:1[di],offset xcpm4
        mov     es:3[di],cs
        mov     xcpm6,sp        ; save sp
        mov     cl,2fh
        int     224     ; program link
xcpm4:
        mov     ax,0040h
        mov     es,ax
        mov     di,0f4ch
        mov     es:word ptr [di],06c6h
        mov     es:word ptr 2[di],2498h
        mov     es:byte ptr 4[di],00
        mov     sp,xcpm6
        mov     ax,cs
        mov     ss,ax
        pop     ds
        pop     es
        retnil
xcpm2   rs      40
xcpm6   dw      0
```

図4「retnil」はコード・マクロになっており「mov ax,nil ! ret」である。普通に利用するには単に「ret」のみでよい

図4の説明を少ししておこう。これは，筆者のLisp05におけるプログラムそのままである。この手順はCP/M-86 V1.1でのみ動作する。しかし，これ以外の版は現在存在していないのでだいたいのCP/M-86で動くはずである。

Lisp05では引数はLispのポインタ形式でbxレジスタに渡される。説明は省略するが「shl bx,1」を2回行うことにより普通の意味での番地になる。

また，そうして得られた番地＋1の所に文字列長，番地＋4の所に文字列へのポインタがあるという形式になっていて，CP/M形式〔最後にnull(0_H：ゼロ値)が来る所までが一つの文字列〕とは少々異なっている。またコマンド・インタプリタに与える文字列は大文字でなければならないので，その変換をあわせて行いながらコマンド・バッファへ文字列をセットする。これがラベルxcpm3近辺のループである。

その後，DMAセグメントの設定とDMAアドレスの設定を行い，CP/M-86常駐部のBDOSの一部とコマンド解釈ループの一部とに変更を加え，BDOS #47を実行する(**図5**参照)。実行を終えて戻ってきたら変更した所を元に戻してやる。

exec機能の利用方法

普通のプログラムでこのコードを利用するために，xcpmxというエントリを設けてある。もしこのコードを利用

するなら，ここから下を利用するとよい。

この場合，ラベル xcpmx への呼び出し規則は，

si レジスタに文字列の先頭番地

cl レジスタに文字列の長さ

を与えて，

call xcpmx

と呼ぶ。レジスタ（ax～dx，si，di）はセグメント・レジスタを除いてまったく保存していない。これは筆者の経験的な規則であるので，必要なレジスタがあれば各自保存してほしい。

利用にあたっての問題点・考慮点

最後に，この CP/M-86 への exec 機能組み込み上の問題点を述べる。

①組み込みコマンド（dir，type 等）を実行するとコマンド・プロンプトに戻ってしまう。鍵盤でコントロールＣを入力すれば制御を回復できるが，それが困難な場合には問題となる。

② BDOS #47 のプログラム・チェインを利用しているので，この exec 機能により実行させたプログラムが SUBMIT コマンドである場合，あるいは，プログラム・チェインを行う場合，そして，

③自分自身と同じメモリ・イメージを呼び出し，そこでさらに exec 呼出しを行う場合，

などに不具合が生じる。

②に対しては，SUBMIT であれば，内容に対応する解析を行い，順にプログラムを起動してやればできないことはない。チェインを使っているものを起動する場合もその流れが事前にわかっているのならば対応する順序で xcpmx を呼び出せばできないことはない。たとえばマルチパス方式の C コンパイラなどがこれに該当する。

なお，余談になるが，**図 4** に示したプログラムを含めて筆者のプログラムは Digital Research の RASM86 というリロケータブル・アセンブラと LIB86，LINK86，(そして一部 DRC)，を用いて作成している。RASM86 にはいわゆるマクロ機能はないが，分割アセンブルできるので少なくとも ASM86 より便利である。また，MS-DOS の MASM を含めたインテル系アセンブラに比べて「鉛筆が減らない」ので，結構重宝している。

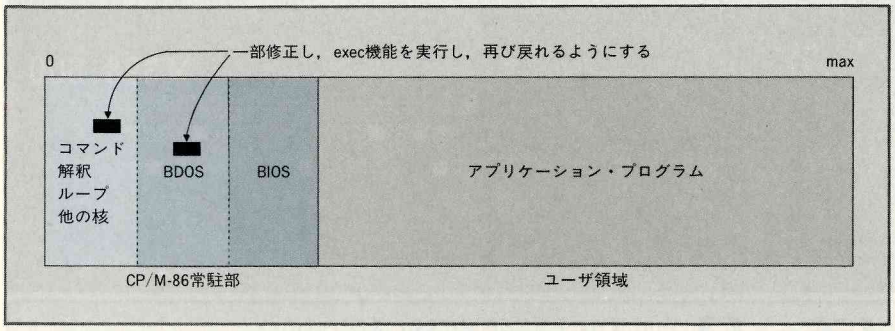

図 5　CP/M-86下でのexec機能の実現に必要な常駐部の一時的修正