МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе № 6

по дисциплине «Операционные системы»

Тема: «Построение модуля динамической структуры»

Студент гр. 8304 _____ Сергеев А.Д.

Преподаватель _____ Губкин А.Ф.

Санкт-Петербург
2020

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры. В отличии от предыдущих лабораторных работ в этой работе рассматривается приложение, состоящее из нескольких модулей, а не из одного модуля простой структуры. В этом случае разумно предположить, что все модули приложения находятся в одном каталоге и полный путь в этот каталог можно взять из среды, как это делалось в работе 2. Понятно, что такое приложение должно запускаться в соответствии со стандартами ОС.

В работе исследуется интерфейс между вызывающим и вызываемым модулями по управлению и по данным. Для запуска вызываемого модуля используется функция 4B00h прерывания int 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Постановка задачи:

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции:

1. Подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится он сам. Вызываемому модулю передается новая среда, созданная вызывающим модулем и новая командная строка.
2. Вызываемый модуль запускается с использованием загрузчика.
3. После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы. Необходимо проверять причину завершения и, в зависимости от значения, выводить соответствующее сообщение. Если причина завершения 0, то выводится код завершения.

В качестве вызываемой программы необходимо взять программу ЛР 2, которая распечатывает среду и командную строку. Эту программу следует

немного модифицировать, вставив перед выходом из нее обращение к функции ввода символа с клавиатуры. Введенное значение записывается в регистр AL и затем происходит обращение к функции выхода 4Ch прерывания int 21h.

Шаг 2. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите произвольный символ из числа A-Z. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

Шаг 3. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите комбинацию символов Ctrl+C Посмотрите причину завершения и код. Занесите полученные данные в отчет.

Шаг 4. Запустите отлаженную программу, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

Повторите ввод комбинаций клавиш. Занесите полученные данные в отчет.

Шаг 5. Запустите отлаженную программу, когда модули находятся в разных каталогах. Занесите полученные данные в отчет.

Необходимые сведения для составления программы.

Для загрузки и выполнения одной программы из другой используется функция 4B00h прерывания int 21h (загрузчик ОС). Перед обращением к этой функции необходимо выполнить следующие действия:

1. Подготовить место в памяти. При начальном запуске программы ей отводится вся доступная в данный момент память 0S, поэтому необходимо освободить место в памяти. Для этого можно использовать функцию 4Ah прерывания int 21h. Эта функция позволяет уменьшить

отведенный программе блок памяти. Перед вызовом функции надо определить объем памяти, необходимый программе ЛР 6 и задать в регистре BX число параграфов, которые будут выделяться программе. Если функция 4Ah не может быть выполнена, то устанавливается флаг переноса CP=1 и в AX заносится код ошибки:

7 — разрушен управляющий блок памяти;

8 — недостаточно памяти для выполнения функции;

9 — неверный адрес блока памяти;

Поэтому после выполнения каждого прерывания int 21h следует проверять флаг переноса CP=1.

2. Создать блок параметров. Блок параметров - это 14-байтовый блок памяти, в который помещается следующая информация:

dw — сегментный адрес среды

dd — сегмент и смещение командной строки

dd — сегмент и смещение первого FCB

dd — сегмент и смещение второго FCB

Если сегментный адрес среды 0, то вызываемая программа наследует среду вызывающей программы. В противном случае вызывающая программа должна сформировать область памяти в качестве среды, начинающуюся с адреса кратного 16 и поместить этот адрес в блок параметров.

Командная строка записывается в следующем формате:

Первый байт — счетчик, содержащий число символов в командной строке, затем сама командная строка, содержащая не более 128 символов.

На блок параметров перед загрузкой вызываемой программы должны указывать ES:BX.

3. Подготовить строку, содержащую путь и имя вызываемой программы. В конце строки должен стоять код ASCII 0. На подготовленную строку должны указывать DS:DX.

4. Сохранить содержимое регистров SS и SP в переменных. При восстановлении SS и SP нужно учитывать, что DS необходимо также восстановить.

Код завершения формируется вызываемой программой в регистре AL перед выходом в OS с помощью функции 4Ch прерывания int 21h.

В качестве вызываемой программы целесообразно использовать программу, разработанную в Лабораторной работе №2, модифицировав ее следующим образом. Перед выходом из программы перед выполнением функции 4Ch прерывания int 21h следует запросить с клавиатуры символ и поместить введенный символ в регистр AL, в качестве кода завершения. Это можно сделать с помощью функции 01h прерывания int 21h.

Введенный символ остается в регистре AL и служит аргументом для функции 4Ch прерывания int 21h.

Описание программы.

В результате выполнения лабораторной работы была написана программа, описание функций которой представлено ниже.

- HEX_BYTE_PRINT - выводит на экран один байт в 16 системе счисления, принимая его в регистре AL;
- PRINT_STRING - вывод строки из DX на экран;
- FREE_MEM - освобождение незанятого прграммой места в памяти;
- FINISH - вывод сообщения на экран в зависимости от завершения вложенной программы;
- FORM_PATHS - формирование пути к текущему каталогу и пути к исполняемому модулю;

Ход работы

Написание исходного кода производилось в редакторе Atom на базе операционной системы Windows 10, сборка и отладка производились в эмуляторе DOSBox. Поскольку эмуляторы DOS не поддерживают прерывания по Ctrl+Break, отладка обработки этого прерывания проводилась в системе Windows XP.

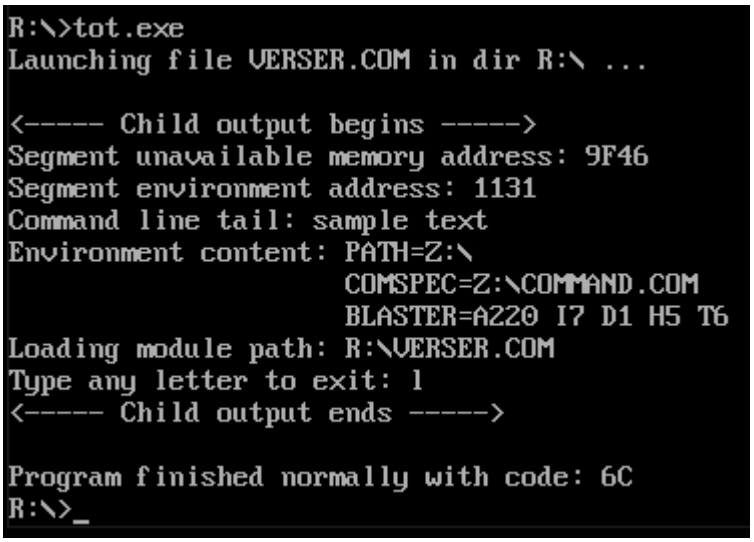

Рисунок 1 — Выполнение программы с прерыванием по введённой букве «1»

Как видно из рисунка, программа завершилась в штатном режиме с кодом 6C.

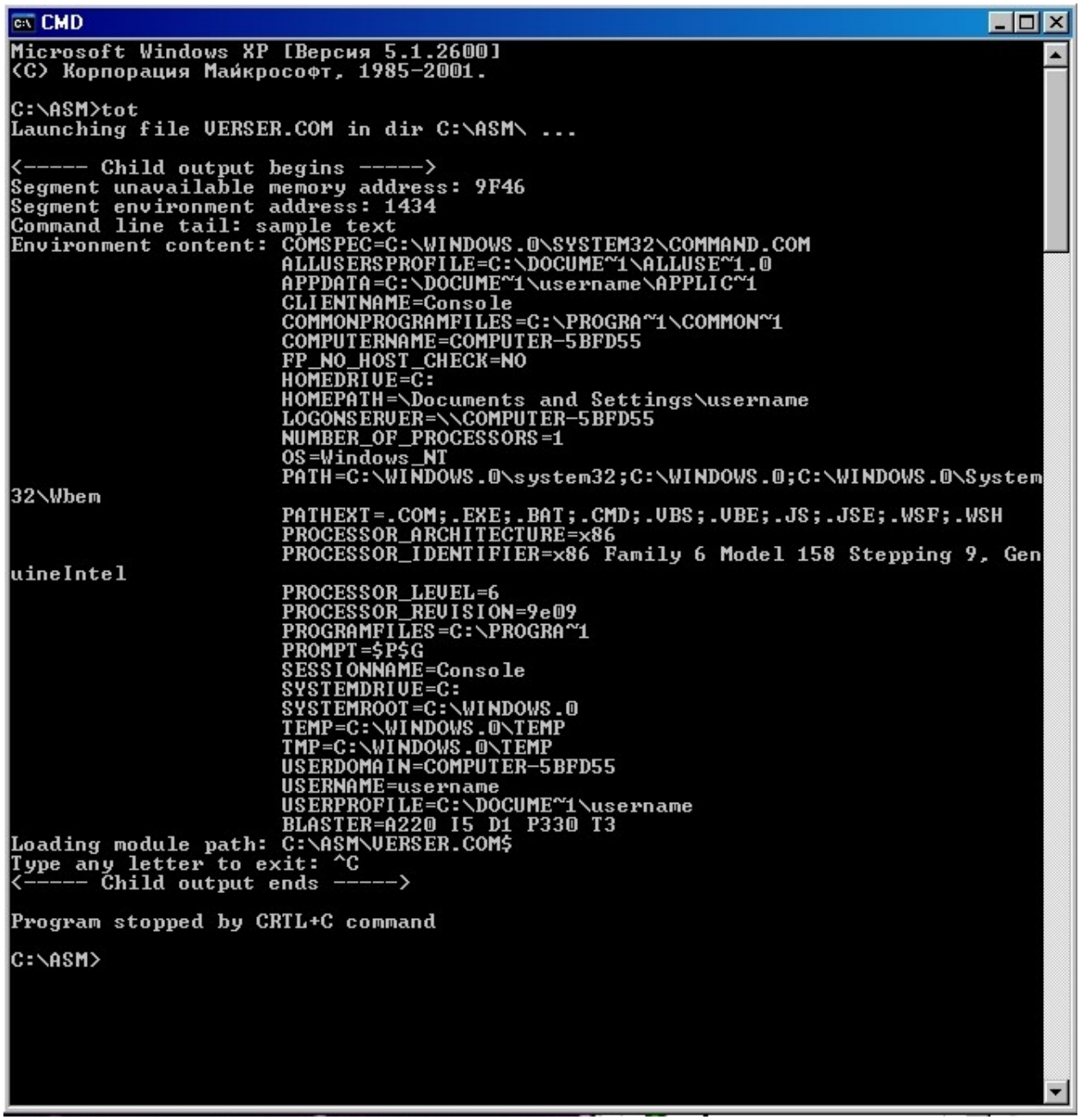

Рисунок 2 – Выполнение программы с прерыванием по введённой комбинации «Ctrl+C»

Рисунок 3 — Программа и модули находятся в разных каталогах

Как видно из рисунка, модуль найден не был, программа verser.exe не была запущена.

Вывод.

В результате выполнения данной лабораторной работы была изучена возможность встраивания пользовательского обработчика прерываний от клавиатуры в стандартный.

Контрольные вопросы.

Как реализовано прерывание Ctrl+C:

Когда нажата комбинация Ctrl+C, DOS вызывает прерывание int 23h. Уровень чувствительности к этому прерыванию может быть проверен и установлен функцией 33h прерывания int 21h.

В какой точке заканчивается вызываемая программа, если код завершения 0:

В месте вызова функции 4Ch прерывания int 21h.

В какой точке заканчивается вызываемая программа по прерыванию Ctrl+Break:

В точке вызова функции 01h прерывания int 21h, гда ожидался ввод из клавиатуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. TOT.ASM

```
ASSUME  CS:CODE,    DS:DATA,    ss:ASTACK


ASTACK SEGMENT STACK

    dw  128 dup(0)

ASTACK ENDS


DATA SEGMENT


        COMP_SUCCESS    db   "Program  finished  normally  with
code: $"

        COMP_CC             db    "Program   stopped   by   CRTL+C
command", 13, 10, "$"

        ERR_FILENAME    db   "File  not  found!",  13,  10,  "Path
searched: $"

         ERR_MEMORY       db  "Not sufficient memory!", 13, 10,
"$"


        START_STR_NAME  db "Launching file $"

        START_STR_PATH  db " in dir $"

        START_STR_END   db " ...", 13, 10, "$"


         DELIM_START      db 13, 10, "<----- Child output begins
----->", 13, 10, "$"
```

```
          DELIM_END        db "<----- Child output ends ----->",
13, 10, 13, 10, "$"


          PATH           db    100 dup(0)

          FILENAME        db   "VERSER.COM$"

          MODULE          db  150 dup(0)


          PSP             dw   ?

          CMD             db  12, "sample text", 0

          KEEP_SS         dw   ?

          KEEP_SP         dw   ?


          PARAMS               dw   7 dup(?)

          MEMORY_ERROR    db   0


     DATA ENDS


     CODE SEGMENT


     HEX_BYTE_PRINT PROC

               push      ax

               push      bx

               push      dx
```

```
        mov  ah, 0
        mov  bl, 10h
        div  bl
        mov  dx, ax
        mov  ah, 02h
        cmp  dl, 0Ah
        jl        PRINT
        add  dl, 07h
    PRINT:
        add  dl, '0'
        int  21h;

        mov  dl, dh
        cmp  dl, 0Ah
        jl        PRINT_EXT
        add  dl, 07h
    PRINT_EXT:
        add  dl, '0'
        int  21h;

        pop  dx
        pop  bx
        pop  ax
       ret
```

```
HEX_BYTE_PRINT ENDP


PRINT_STRING   PROC near
          push      ax
          mov  ah, 09h
          int       21h
          pop  ax
         ret
PRINT_STRING   ENDP




FREE_MEM  PROC
          mov  bx, offset PROGEND
          mov  ax, es
          sub  bx, ax
          mov  cl, 4
          shr  bx, cl
          mov  ah, 4Ah
          int  21h
          jc        MCATCH
          jmp  MDEFAULT
     MCATCH:
          mov  MEMORY_ERROR, 1
```

```
     MDEFAULT:
          ret
FREE_MEM  ENDP



FINISH    PROC
          mov  ah, 4Dh
          int  21h
          cmp  ah, 1
          je        ECTRLC
          mov  dx, offset COMP_SUCCESS
          call      PRINT_STRING
          call HEX_BYTE_PRINT
          mov  dl, ah
          mov  ah, 2h
          int  21h
          jmp  EDEFAULT
     ECTRLC:
          mov  dx, offset COMP_CC
          call      PRINT_STRING
     EDEFAULT:
          ret
FINISH    ENDP
```

```
FORM_PATHS          PROC near
          push    ax
          push    es
          push    si
        push    di
          push    dx

          mov     ax, es:[2Ch]
        mov     es, ax

          mov     si, 0
SYMBOL:
        add     si, 1
          mov     al, es:[si]
          mov     ah, es:[si+1]
          cmp     ax, 0
          jne     SYMBOL

        add     si, 4
        mov     di, 0
SYMB:
```

```
        mov     al, es:[si]
        cmp     al, 0
        je      DIR_END

        mov     PATH[di], al

        inc     di
        add     si, 1
        jmp     SYMB

DIR_END:
        cmp     PATH[di - 1], "\"
        je      APPENSION
        mov     PATH[di - 1], "$"
        dec     di
        jmp     DIR_END

APPENSION:
        mov     PATH[di], "$"

        mov     si, 0
        mov     di, 0
ADD_PATH:
        cmp     PATH[di], "$"
```

```
        je      ADD_NAME_PREP
        mov     al, PATH[di]
        mov     MODULE[si], al
        inc     si
        inc     di
        jmp     ADD_PATH

ADD_NAME_PREP:
        mov     di, 0
ADD_NAME:
        cmp     FILENAME[di], "$"
        je      P_END
        mov     al, FILENAME[di]
        mov     MODULE[si], al
        inc     si
        inc     di
        jmp     ADD_NAME

P_END:
        mov     MODULE[si], "$"

          pop     dx
        pop     di
          pop     si
```

```
          pop     es
          pop     ax
          ret
FORM_PATHS          ENDP



MAIN PROC
          mov  ax, DATA
          mov  ds, ax

        ; -----> Forming path to module
        call    FORM_PATHS

        mov     dx, offset START_STR_NAME
        call    PRINT_STRING
        mov     dx, offset FILENAME
        call    PRINT_STRING
        mov     dx, offset START_STR_PATH
        call    PRINT_STRING
        mov     dx, offset PATH
        call    PRINT_STRING
        mov     dx, offset START_STR_END
        call    PRINT_STRING
```

```
; -----> Freeing memory
  call      FREE_MEM
  cmp  MEMORY_ERROR, 0
  jne  PDEFAULT

; -----> Creating Environment
mov     PARAMS[0], 0

; -----> Preparing cmd
mov     PARAMS[2], offset CMD
mov     PARAMS[4], ds

mov     dx, offset DELIM_START
call    PRINT_STRING

  push      ds
  pop  es

mov     ax, ds
mov     es, ax
  mov  dx, offset MODULE
  mov  bx, offset PARAMS
```

```
      mov  KEEP_SS, ss
      mov  KEEP_SP, sp

      mov  ax, 4B00h
      int  21h

    mov     cx, DATA
    mov     ds, cx

      mov  ss, KEEP_SS
      mov  sp, KEEP_SP

    mov     dx, offset DELIM_END
    call    PRINT_STRING

    ; -----> Printing results
      jc        NOFILE
      jmp  SUCCESS

NOFILE:
      mov  dx, offset ERR_FILENAME
      call      PRINT_STRING
      mov  dx, offset FILENAME
      call      PRINT_STRING
```

```
            jmp  PDEFAULT


      SUCCESS:

            call      FINISH


      PDEFAULT:

            mov  ah, 4Ch

            int  21h

MAIN ENDP

PROGEND:


CODE ENDS


end MAIN
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. VERSER.COM

```
VERSER SEGMENT

            ASSUME    CS:VERSER,    DS:VERSER,    ES:NOTHING,
SS:NOTHING

            org 100H

START:    jmp  BEGIN


UNAVAILABLE_ADDRESS          db   'Segment      unavailable
memory address: $'

ENVIRONMENT_ADDRESS          db    'Segment      environment
address: $'

COMMAND_LINE_TAIL            db   'Command line tail: $'

ENVIRONMENT_CONTENT          db    'Environment content: $'

LOADING_PATH             db   'Loading module path: $'


END_STRING                   db   'Type   any   letter   to
exit: $'


TWO_SYMBOLS_CONTAINER        db   '  $'

ENVIRONMENTAL_LINE           db    0dh,        0ah,        '
$'

ENDL_LINE                    db   0dh, 0ah, '$'
```

```
PRINT_STRING   PROC near
          push      ax
          mov  ah, 09h
          int       21h
          pop  ax
          ret
PRINT_STRING   ENDP



;-------------------------------------------------------------
----------------------



TETR_TO_HEX         PROC near
          and       al, 0fh
          cmp       al, 09
          jbe       NEXT
          add       al, 07
NEXT:     add       al, 30h
          ret
TETR_TO_HEX         ENDP
```

```
    ;-------------------------------------------------------
----------------------


    BYTE_IN_HEX         PROC      near
                push cx
                push      si
                mov       al, ah
                call TETR_TO_HEX
                xchg al, ah
                mov       cl, 4
                shr       al, cl
                call TETR_TO_HEX
                lea  si, TWO_SYMBOLS_CONTAINER
                mov  [si], al
                mov  [si+1], ah
                mov  dx, offset TWO_SYMBOLS_CONTAINER
                call      PRINT_STRING
                pop  si
                pop       cx
                ret
    BYTE_IN_HEX         ENDP


    ;-------------------------------------------------------
----------------------
```

```
PRINT_REG      PROC      near
               push cx
               push      dx
               mov  cx, ax
               call BYTE_IN_HEX
               mov  al, ch
               call BYTE_IN_HEX
               mov  dx, offset ENDL_LINE
               call      PRINT_STRING
               pop       dx
               pop       cx
               ret
PRINT_REG      ENDP




     ;-------------------------------------------------------------
----------------------




PRINT_TAIL          PROC near
               push ax
```

```
                push cx
                push si
                mov cx, 0
                mov cl, ds:[80h]
                cmp CL, 0
                je PT_END
                mov si, 0
                mov ax, 0
CYCLE:          mov al, ds:[81h+si]
                call PRINT_BYTE
                add si, 1
                loop CYCLE
PT_END:         mov dx, offset ENDL_LINE
                call PRINT_STRING
                pop si
                pop cx
                pop ax
                ret
PRINT_TAIL      ENDP

;-------------------------------------------------------------
----------------------

PRINT_BYTE      PROC near
```

```
                push ax
                push dx
                mov dx, 0h
                mov dl, al
                mov ah, 02h
                int 21h
                pop dx
                pop ax
                ret
    PRINT_BYTE          ENDP

    ;-------------------------------------------------------------
----------------------


    PRINT_ENV      PROC near
                push ax
                push es
                push si
                push dx
                push bx
                mov bx, 2Ch
                mov es, [bx]
                pop bx
                mov si, 0
```

```
                mov al, es:[si]
SYMBOL:     cmp al, 0
                jne NOLINE
                mov dx, offset ENVIRONMENTAL_LINE
                call PRINT_STRING
                jmp LOOPER
NOLINE:     call PRINT_BYTE
LOOPER:     add si, 1
                mov al, es:[si]
                mov ah, es:[si+1]
                cmp ax, 0
                jne SYMBOL
                mov dx, offset ENDL_LINE
                call PRINT_STRING
                call PRINT_PATH
                pop dx
                pop si
                pop es
                pop ax
                ret
PRINT_ENV       ENDP

;-------------------------------------------------------------
----------------------
```

```
PRINT_PATH          PROC near
          mov dx, offset LOADING_PATH
          call PRINT_STRING
          add si, 4
SYMB:     mov al, es:[si]
          cmp al, 0
          je P_END
          call PRINT_BYTE
          add si, 1
          jmp SYMB
P_END:    ret
PRINT_PATH          ENDP



;-------------------------------------------------------------
----------------------



BEGIN:
          push    dx
          push      ax
```

```
mov  dx, offset UNAVAILABLE_ADDRESS
call      PRINT_STRING
mov  ax, ds:[02h]
call PRINT_REG

mov  dx, offset ENVIRONMENT_ADDRESS
call      PRINT_STRING
mov  ax, ds:[2Ch]
call PRINT_REG

mov  dx, offset COMMAND_LINE_TAIL
call      PRINT_STRING
call PRINT_TAIL

mov  dx, offset ENVIRONMENT_CONTENT
call      PRINT_STRING
call PRINT_ENV

mov  dx, offset ENDL_LINE
call      PRINT_STRING

mov  dx, offset END_STRING
call      PRINT_STRING
```

```
ENDING:
          pop       ax
          pop  dx


          mov  ah, 01h
          int  21h
          mov  dx, offset ENDL_LINE
          call      PRINT_STRING
          mov  ah, 4ch
          int       21h
          ret
VERSER    ENDS
          END       START
```