

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

ОТЧЕТ
по лабораторной работе №6
по дисциплине «Операционные системы»
Тема: Построение модуля динамической структуры

Студент гр. 8383

Ларин А.

Преподаватель

Ефремов М.А,

Санкт-Петербург


2020

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры.

Выполнение

1. Написан код .EXE модуля, который выполняет следующие функции:
 - 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.
 - Вызываемый модуль запускается с использованием загрузчика.
 - 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
2. Программа запущена, когда текущим каталогом является каталог с разработанными модулями. Для проверки вводится произвольный символ из A-Z. Результат приведен на рис. 1

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX - 

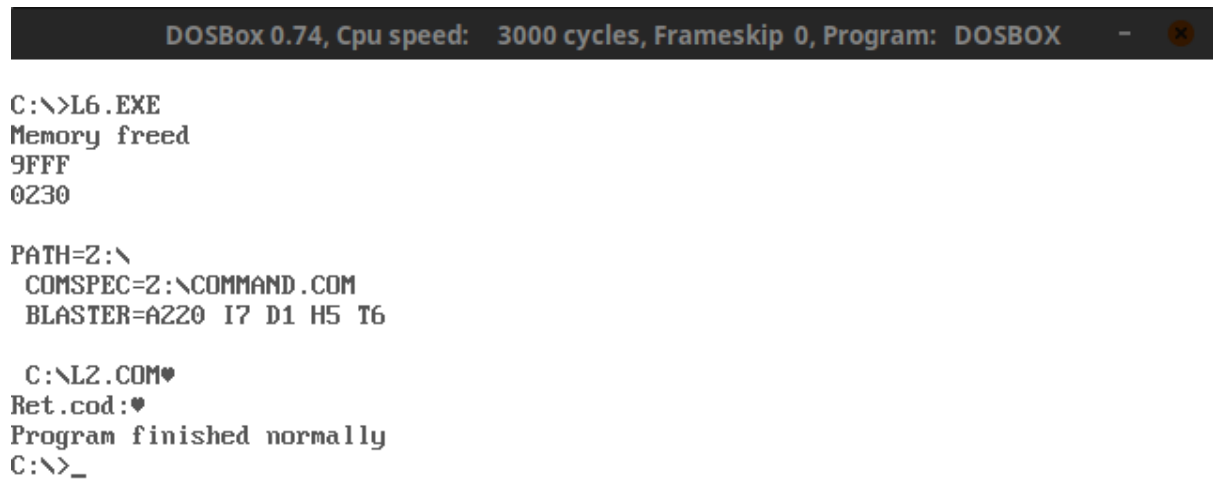
C:\>L6.EXE
Memory freed
9FFF
0230

PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

C:\>L2.COMQ
Ret.cod:Q
Program finished normally
C:\>_
```

Рисунок 1 – Запуск программы из каталога с модулями, введен символ Q

3. Программа запущена, когда текущим каталогом является каталог с разработанными модулями. Для проверки вводится комбинация Ctrl+C. Результат приведен на рис. 2



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX - X

C:\>L6.EXE
Memory freed
9FFF
0230

PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

C:\>L2.COM♥
Ret.cod:♥
Program finished normally
C:\>_
```

Рисунок 2 – Запуск программы из каталога с модулями, введена комбинация Ctrl+C

4. Программа запущена, когда текущим является каталог отличный от каталога с модулями является каталог с разработанными модулями. Для проверки введены те-же комбинации. Результаты представлены на рис. 3 и рис. 4

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

C:\>mkdir ae

C:\>cd ae

C:\AE>dir
Directory of C:\AE\
.                <DIR>                17-05-2020 21:34
..               <DIR>                17-05-2020 21:34
    0 File(s)                0 Bytes.
    2 Dir(s)                262,111,744 Bytes free.

C:\AE>..\L6.EXE
Memory freed
9FFF
0230

PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

C:\L2.COMQ
Ret.cod:Q
Program finished normally
C:\AE>
```

Рисунок 3 – Запуск программы из другого каталога, введен символ Q

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

C:\>cd AE

C:\AE>..\L6.EXE
Memory freed
9FFF
0209

PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

C:\L2.COM♥
Ret.cod:♥
Program finished normally
C:\AE>
```

Рисунок 4 – Запуск программы из другого каталога, введена комбинация Ctrl+C

5. Программа запущена, когда модули находятся в разных каталогах. Результат представлен на рис. 5

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX - X
C:\>L6.EXE
Memory freed
Failed to load program
LOAD_ERR: File not found
C:\>_
```

Рисунок 5 – Запуск программы с модулями в разных каталогах

Контрольные вопросы

1. Как реализовано прерывание Ctrl-C?

По нажатию Ctrl-C в буфер клавиатуры поступает код 03. После его обнаружения осуществляется вызов прерывания 23h, обработчик которого завершает текущий процесс, передавая управление родительскому с кодом причины завершения 1.

2. В какой точке заканчивается вызываемая программа, если код причины завершения 0?

В точке, где происходит завершение программы с кодом, например по функции 4Ch прерывания 21h

3. В какой точке заканчивается вызываемая программа по прерыванию Ctrl-C?

Вызываемая программы завершается в момент обнаружения кода комбинации Ctrl-C DOS'ом. При флаге BREAK=ON его распознавание

происходит при вызове любой из функций DOS кроме 06 и 07. При BRAKE=OFF он распознается при операции ввода/вывода в консоль.

Выводы.

В результате работы были разобраны некоторые концепции языка ассемблера и работы операционной системы DOS. Были исследованы способы загрузки резидентной программы и установки своего обработчика прерываний. Написан модуль EXE с требуемым функционалом.

ПРИЛОЖЕНИЕ А

L6.ASM

DATA SEGMENT

IS_MEM_FREED db 0

KEEP_PSP dw 0

CLN dw 0

PATH db 80h dup(0)

PROG_NAME db "L2.COM", 0

KEEP_SP dw 0

KEEP_SS dw 0

STR_RET_CODE db 13, 10, "Ret.cod: \$",0

STR_NEW_LINE db 0DH,0AH,'\$'

STR_MEM_ERROR db 'Failed to free memory\$'

STR_MEM_SUCCESS db 'Memory freed\$'

STR_MEM_ERROR7 db 'MEM_ERR: MCB corrupted\$'

STR_MEM_ERROR8 db 'MEM_ERR: Not enough memory to
execute\$'

STR_MEM_ERROR9 db 'MEM_ERR: Invalid memory block adress\$'

STR_LOAD_ERROR db 'Failed to load program\$'

STR_LOAD_SUCCESS db 'Program loaded\$'

STR_LOAD_ERROR1 db 'LOAD_ERR: Invalid function nunmber\$'

STR_LOAD_ERROR2 db 'LOAD_ERR: File not found\$'

STR_LOAD_ERROR5 db 'LOAD_ERR: disk error\$'

STR_LOAD_ERROR8 db 'LOAD_ERR: Not enough memory\$'

STR_LOAD_ERROR10 db 'LOAD_ERR: Invalid environment line\$'

STR_LOAD_ERROR11 db 'LOAD_ERR: Wrong format\$'

STR_END_CAUSE0 db 'Program finished normally\$'

STR_END_CAUSE1 db 'Program finished by Ctrl+Break\$'

STR_END_CAUSE2 db 'Program finished by device error\$'

STR_END_CAUSE3 db 'Program went resident\$'

PARAMS dw ? ;сегментный адрес среды

dd ? ;сегмент и смещение командной строки

dd ? ;сегмент и смещение первого FCB

dd ? ;сегмент и смещение второго FCB

DATA_END db 0

DATA ENDS

PSTACK SEGMENT STACK

dw 128 dup(0)

PSTACK ENDS

```

CODE SEGMENT
    assume cs:CODE, ds:DATA, ss:PSTACK, es:NOTHING
;=====
FREE_MEM proc near
    push ax
    push bx
    push cx
    push dx
    push es

    mov ax,offset DATA_END
    mov bx,offset MEM_END
    add bx,ax
    add bx,30Fh
    mov cx,4
    shr bx,cl
    xor al,al
    mov ah,4Ah
    int 21h

    jnc MEM_FREED
    mov dx,offset STR_MEM_ERROR
    call PRINT
    call LN
    mov IS_MEM_FREED,0

    cmp ax, 7
        je MEM_ERROR7
    cmp ax, 8
        je MEM_ERROR8
    cmp ax, 9
        je MEM_ERROR9

MEM_ERROR7:
    mov dx, offset STR_MEM_ERROR7
    jmp FREE_MEM_END
MEM_ERROR8:
    mov dx, offset STR_MEM_ERROR8
    jmp FREE_MEM_END
MEM_ERROR9:
    mov dx, offset STR_MEM_ERROR9
    jmp FREE_MEM_END

MEM_FREED:
    mov IS_MEM_FREED,1
    mov dx, offset STR_MEM_SUCCESS
FREE_MEM_END:
    call PRINT

```



```

    call LN

    pop es
    pop dx
    pop cx
    pop bx
    pop ax
    ret
FREE_MEM endp

;=====
PROCESS proc near

    push ax
    push cx
    push dx
    push si
    push di
    push es

    mov ax, KEEP_PSP
    mov es, ax
    mov es, es:[2Ch]
    mov si, 0

SEEK_START:
    mov ax, es:[si]
    inc si
    cmp ax, 0
    jne SEEK_START
    add si, 3
    mov di, 0
SEEK_CLN:
    mov al, es:[si]
    cmp al, 0
    je APPEND_NAME
    cmp al, '\'
    jne ADD_PATH_SYM
    mov CLN, di
ADD_PATH_SYM:
    mov BYTE PTR [PATH + di], al
    inc si
    inc di
    jmp SEEK_CLN
APPEND_NAME:
    mov di, CLN
    inc di

```

```

    add di, offset PATH
    mov si, offset PROG_NAME
    mov ax, ds
    mov es, ax
APPEND_SYMB:
    mov al, ds:[si]
    mov ds:[di], al
    inc di
    inc si
    cmp al, 0
    jne APPEND_SYMB

    pop dx
    pop si
    pop di
    pop es
    pop cx
    pop ax
    ret
PROCESS endp
;=====
LOAD proc near
    push ax
    push bx
    push dx

    push ds
    push es
    mov KEEP_SP, SP
    mov KEEP_SS, ss

    mov ax, DATA
    mov es, ax
    mov bx, offset PARAMS
    mov dx, offset CLN
    mov [bx + 2], dx
    mov [bx + 4], ds
    mov dx, offset PATH
    mov ax, 4B00h
    int 21h

    mov ss, ds:KEEP_SS
    mov SP, ds:KEEP_SP
    pop es
    pop ds

    jnc LOAD_SUCCESS

```

```

    mov dx, offset STR_LOAD_ERROR
    call PRINT
    call LN
    cmp ax, 1
    je LOAD_ERROR1
    cmp ax, 2
    je LOAD_ERROR2
    cmp ax, 5
    je LOAD_ERROR5
    cmp ax, 8
    je LOAD_ERROR8
    cmp ax, 10
    je LOAD_ERROR10
    cmp ax, 11
    je LOAD_ERROR11
LOAD_ERROR1:
    mov dx, offset STR_LOAD_ERROR1
    call PRINT
    call LN
    jmp LOAD_END
LOAD_ERROR2:
    mov dx, offset STR_LOAD_ERROR2
    call PRINT
    call LN
    jmp LOAD_END
LOAD_ERROR5:
    mov dx, offset STR_LOAD_ERROR5
    call PRINT
    call LN
    jmp LOAD_END
LOAD_ERROR8:
    mov dx, offset STR_LOAD_ERROR8
    call PRINT
    call LN
    jmp LOAD_END
LOAD_ERROR10:
    mov dx, offset STR_LOAD_ERROR10
    call PRINT
    call LN
    jmp LOAD_END
LOAD_ERROR11:
    mov dx, offset STR_LOAD_ERROR11
    call PRINT
    call LN
    jmp LOAD_END
LOAD_SUCCESS:
    mov ax, 4D00h
    int 21h

```

```

    mov di, offset STR_RET_CODE
    mov [di + 10], al
    mov dx, offset STR_RET_CODE
    call PRINT
    call LN
    cmp ah, 0
    je LOAD_END0
    cmp ah, 1
    je LOAD_END1
    cmp ah, 2
    je LOAD_END2
    cmp ah, 3
    je LOAD_END3
LOAD_END0:
    mov dx, offset STR_END_CAUSE0
    call PRINT
    jmp LOAD_END
LOAD_END1:
    mov dx, offset STR_END_CAUSE1
    call PRINT
    jmp LOAD_END
LOAD_END2:
    mov dx, offset STR_END_CAUSE2
    call PRINT
    jmp LOAD_END
LOAD_END3:
    mov dx, offset STR_END_CAUSE3
    call PRINT

LOAD_END:
    pop dx
    pop bx
    pop ax
    ret
LOAD endp
;=====
;UTILS
;=====
PRINT PROC NEAR
    PUSH ax
    MOV AH, 09H
    INT 21H
    POP ax
    RET
PRINT ENDP
;=====
LN PROC

```

```

    push ax
    push dx
    mov dx, offset STR_NEW_LINE
    mov ah, 9h
    int 21h
    pop dx
    pop ax
    ret
LN ENDP
;=====
PRINTLN PROC NEAR
    call PRINT
    call LN
    RET
PRINTLN ENDP

;=====MAIN=====MAIN=====MAIN=====MAIN=====
MAIN PROC
;MAIN init
    mov ax, DATA ;ds setup
    mov ds, ax
    mov KEEP_PSP, es

    call FREE_MEM

    cmp IS_MEM_FREED, 1
    jne EXIT
    call PROCESS
    call LOAD

EXIT:
    xor al, al
    mov ah, 4Ch
    int 21h

MAIN ENDP

MEM_END:
CODE ENDS

END MAIN

```

ПРИЛОЖЕНИЕ Б

L2.ASM

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
```

```
START:  JMP BEGIN
;data
```

```
NEW_LINE db 0DH,0AH,'$'
```

```
STRING DB 'Some text          ',0DH,0AH,'$'
;procedures
```

```
DIGIT_TO_CHAR PROC near
```

```
;AL
    and al,0Fh
    cmp al,09h
    jle BLW
    add al,'A'
    sub al, 0Ah
    jmp DTC_CONT
```

```
BLW:
    add al,'0'
```

```
DTC_CONT:
    ret
```

```
DIGIT_TO_CHAR ENDP
```

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

```
PRINT_AS_HEX proc near
```

```
;AL - number
;;breaks AX,CX,BX
    push ax
    push bx
    push cx
    push dx
    ;mov bx,dx
    mov ch,al
    mov cl,4
    shr al,cl
    call DIGIT_TO_CHAR
    mov dl,al
    mov ah,02h
    int 21h
    mov al,ch
```

```

    call DIGIT_TO_CHAR
    mov dl,al
    mov ah,02h
    int 21h
    ;mov dx,bx
    pop dx
    pop cx
    pop bx
    pop ax

    ret
PRINT_AS_HEX ENDP

PRINT_WORD proc near
;AX - word
    xchg AL,AH
    call PRINT_AS_HEX
    xchg AL,AH
    call PRINT_AS_HEX
    ret
PRINT_WORD ENDP

LN PROC
    push AX
    push DX
    mov DX, offset NEW_LINE
    mov AH, 9h
    int 21h
    pop DX
    pop AX
    ret
LN ENDP

;-----
BEGIN:
;Unaccessible memory
    mov AX, DS:[02h]
    call PRINT_WORD

    call LN
;Enviroment
    mov AX, DS:[2Ch]
    call PRINT_WORD

call LN

```

```

;Tail
    xor CX,CX
    mov CL, DS:[80h]
    xor SI, SI
lp: mov DL,[81h + SI]
    mov AH,02h
    int 21h
    inc DX
    loop lp

    call LN
;Enviroment content
    mov BX, 2Ch
    mov ES, [BX]
    xor SI,SI
    xor AX,AX
lp1:
    mov AX,ES:[SI]
    cmp AX,0001h
    je env_end
    cmp AL,0
    jne cnt
    call LN
cnt:
    mov DL,AL
    xor AX,AX
    mov AH,02h
    int 21h
    inc SI
    loop lp1

env_end:
    add SI,2
lp2:
    mov AX,ES:[SI]
    cmp AL,0
    je path_end
    mov DL,AL
    xor AX,AX
    mov AH,02h
    int 21h
    inc SI
    loop lp2
path_end:

EXIT:
    mov AH, 01h

```



```
int 21h
mov AH,4Ch
int 21h
TESTPC ENDS
END START
```