

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

ОТЧЕТ
по лабораторной работе №7
по дисциплине «Операционные системы»
Тема: Построение модуля оверлейной структуры

Студентка гр. 8381

Преподаватель

Ивлева О.А.

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построение загрузочного модуля оверлейной структуры.

Основные теоретические положения.

Исследуется структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Для запуска вызываемого оверлейного модуля используется функция 4B03h прерывания int 21h. Все загруженные и оверлейные модули находятся в одном каталоге. В этой работе также рассматривается приложение, состоящее из нескольких модулей, поэтому все модули помещаются в один каталог и вызываются с использованием полного пути.

Выполнение работы.

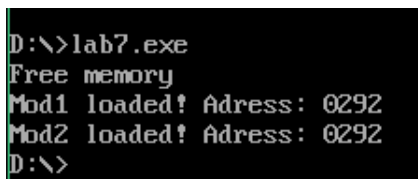
Сборка, отладка производились на базе эмулятора DOSBox 0.74-3.

Был написан текст исходного .EXE модуля с именем lab7.EXE.

Программа выполняет освобождение памяти, выделенной для программы, далее происходит определение размера .OVL файла, если файл не найден, то выводится сообщение, а затем программа завершается.

Если файл найден, то происходит его выполнение. Далее происходит возврат в программу.

Результат выполнения, когда файлы найдены, представлен на рис. 1.



```
D:\>lab7.exe
Free memory
Mod1 loaded! Address: 0292
Mod2 loaded! Address: 0292
D:\>
```

Рисунок 1 – Результат выполнения, когда файлы найдены

Выполнение программы когда один файл не находится в директории и без двух файлов представлен на рис. 2, на рис. 3.

```
D:\>lab7.exe
Free memory
Mod1 loaded! Address: 0292
No OVL size
D:\>
```

Рисунок 2 – Результат выполнения без одного оверлея

```
D:\>lab7.exe
Free memory
No OVL size
No OVL size
D:\>_
```

Рисунок 3 – Результат выполнения без двух оверлеев

Выполнение программы в другой директории, представлен на рис. 4.

```
D:\>cd test
D:\TEST>lab7.exe
Free memory
Mod1 loaded! Address: 0292
Mod2 loaded! Address: 0292
D:\TEST>
```

Рисунок 4 – Результат выполнения в другой директории

Контрольные вопросы.

1. Как должна быть устроена программа, если в качестве оверлейного сегмента использовать .COM модули?

В оверлейных сегментах необходимо сохранять и восстанавливать регистры. Также его нужно вызывать по смещению 100h, так как в COM модулях код располагается с 100h. Если этого не сделать, то PSP не будет сформирован.

Выводы.

В ходе выполнения лабораторной работы была исследована возможность построения загрузочного модуля оверлейной структуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. LR7.ASM

```

DATA SEGMENT
    PSP_SEGMENT dw 0

    OVL_PARAM_SEG          dw 0
    OVL_ADDRESS             dd 0
    MOD1_INFO               db "mod1.ovl", 0
    MOD2_INFO               db "mod2.ovl", 0
    PATH_INFO               db 100h dup(0)
    OFFSET_OVL_NAME         dw 0
    NAME_POS                dw 0
    MEMORY_ERROR            dw 0

    MEMORY_INFO             db "Free memory$"
    ERROR_INFO              db 13, 10, "No OVL size$"
    NOFILE_INFO             db 13, 10, "No OVL file$"
    NOPATH_INFO             db 13, 10, "No OVL path$"
    STR_ERROR_LOAD          db 13, 10, "Error$"

    DTA                     db 43 dup(0)
DATA ENDS

STACKK SEGMENT STACK
    dw 100h dup (0)
STACKK ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:STACKK

WRITE    PROC    near
        push    AX
        mov     AH, 09h
        int     21h
        pop     AX
        ret
WRITE    ENDP

FREEMEM  PROC
        lea     BX, MARK
        mov     AX, ES
        sub     BX, AX
        mov     CL, 8
        shr     BX, CL
        sub     AX, AX
        mov     AH, 4Ah
        int     21h
        jc      CATCH
        mov     DX, offset MEMORY_INFO
        call    WRITE
        jmp     DEFAULT
CATCH:

```

```

        mov     MEMORY_ERROR, 1
DEFAULT:
    ret
FREEMEM     ENDP

```

```

OVLSTART PROC

```

```

    push     AX
    push     BX
    push     CX
    push     DX
    push     SI

    mov     OFFSET_OVL_NAME, AX
    mov     AX, PSP_SEGMENT
    mov     ES, AX
    mov     ES, ES:[2Ch]
    mov     SI, 0
ZERO:
    mov     AX, ES:[SI]
    inc     SI
    cmp     AX, 0
    jne     ZERO
    add     SI, 3
    mov     DI, 0
PATH_START:
    mov     AL, ES:[SI]
    cmp     AL, 0
    je      PATH_START_NAME
    cmp     AL, '\'
    jne     NEW_SYMB
    mov     NAME_POS, DI
NEW_SYMB:
    mov     BYTE PTR [PATH_INFO + DI], AL
    inc     DI
    inc     SI
    jmp     PATH_START
PATH_START_NAME:
    cld
    mov     DI, NAME_POS
    inc     DI
    add     DI, offset PATH_INFO
    mov     SI, OFFSET_OVL_NAME
    mov     AX, DS
    mov     ES, AX
UPDATE:
    lodsb
    stosb
    cmp     AL, 0
    jne     UPDATE

    mov     AX, 1A00h
    mov     DX, offset DTA
    int     21h

```

```

        mov     AH, 4Eh
        mov     CX, 0
        mov     DX, offset PATH_INFO
        int     21h

        jnc     NOERROR
        mov     DX, offset ERROR_INFO
        call    WRITE
        cmp     AX, 2
        je      NOFILE
        cmp     AX, 3
        je      NOPATH
        jmp     PATH_ENDING
NOFILE:
        mov     DX, offset NOFILE_INFO
        call    WRITE
        jmp     PATH_ENDING
NOPATH:
        mov     DX, offset NOPATH_INFO
        call    WRITE
        jmp     PATH_ENDING
NOERROR:
        mov     SI, offset DTA
        add     SI, 1Ah
        mov     BX, [SI]
        mov     AX, [SI + 2]
        mov     CL, 4
        shr     BX, CL
        mov     CL, 12
        shl     AX, CL
        add     BX, AX
        add     BX, 2
        mov     AX, 4800h
        int     21h

        jnc     SETTER
        jmp     PATH_ENDING
SETTER:
        mov     OVL_PARAM_SEG, AX
        mov     DX, offset PATH_INFO
        push    DS
        pop     ES
        mov     BX, offset OVL_PARAM_SEG
        mov     AX, 4B03h
        int     21h

        jnc     LSUCCESS
        mov     DX, offset STR_ERROR_LOAD
        call    WRITE
        jmp     PATH_ENDING
LSUCCESS:
        mov     AX, OVL_PARAM_SEG

```

```

        mov     ES, AX
        mov     WORD PTR OVL_ADRESS + 2, AX
        call    OVL_ADRESS
        mov     ES, AX
        mov     AH, 49h
        int     21h

PATH_ENDING:
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        ret

OVLSTART ENDP

BEGIN:
        mov     AX, DATA
        mov     DS, AX
        mov     PSP_SEGMENT, ES
        call    FREEMEM
        cmp     MEMORY_ERROR, 1
        je      MAIN_END
        mov     AX, offset MOD1_INFO
        call    OVLSTART
        mov     AX, offset MOD2_INFO
        call    OVLSTART

MAIN_END:
        mov     AX, 4C00h
        int     21h

MARK:
CODE ENDS
END BEGIN

```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. MOD1.ASM

```
CODE SEGMENT
    ASSUME CS:CODE, DS:NOTHING, SS:NOTHING
    MAIN PROC FAR
        push AX
        push DX
        push DS

        mov AX, CS
        mov DS, AX
        mov DX, offset STR_RESULT
        call WRITE
        call WRITE_HEX_WORD

        pop DS
        pop DX
        pop AX
        retf
    MAIN ENDP

    STR_RESULT db 13, 10, "Mod1 loaded! Adress: $"

    WRITE PROC
        push AX
        mov AH, 9h
        int 21h
        pop AX
        ret
    WRITE ENDP

    WRITE_HEX_BYTE PROC
        push AX
        push BX
        push DX

        mov AH, 0
        mov BL, 16
        div BL
        mov DX, AX
        mov AH, 02h
        cmp DL, 0Ah
        jl PRINT
        add DL, 7
    PRINT:
        add DL, '0'
```



```

        int 21h;

        mov DL, DH
        cmp DL, 0Ah
        jl PRINT2
        add DL, 7
PRINT2:
        add DL, '0'
        int 21h;

        pop DX
        pop BX
        pop AX
        ret
WRITE_HEX_BYTE ENDP

WRITE_HEX_WORD PROC
        push AX

        push AX
        mov AL, AH
        call WRITE_HEX_BYTE
        pop AX
        call WRITE_HEX_BYTE

        pop AX
        ret
WRITE_HEX_WORD ENDP

CODE ENDS
END MAIN

```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. MOD2.ASM

```
CODE SEGMENT
    ASSUME CS:CODE, DS:NOTHING, SS:NOTHING
    MAIN PROC FAR
        push AX
        push DX
        push DS

        mov AX, CS
        mov DS, AX
        mov DX, offset STR_RESULT
        call WRITE
        call WRITE_HEX_WORD

        pop DS
        pop DX
        pop AX
        retf
    MAIN ENDP

    STR_RESULT db 13, 10, "Mod2 loaded! Adress: $"

    WRITE_HEX_BYTE PROC
        push AX
        push BX
        push DX

        mov AH, 0
        mov BL, 16
        div BL
        mov DX, AX
        mov AH, 02h
        cmp DL, 0Ah
        jl PRINT
        add DL, 7
    PRINT:
        add DL, '0'
        int 21h;

        mov DL, DH
        cmp DL, 0Ah
        jl PRINT2
        add DL, 7
    PRINT2:
        add DL, '0'
```

```

        int 21h;

        pop DX
        pop BX
        pop AX
        ret
WRITE_HEX_BYTE ENDP

WRITE_HEX_WORD PROC
        push AX

        push AX
        mov AL, AH
        call WRITE_HEX_BYTE
        pop AX
        call WRITE_HEX_BYTE

        pop AX
        ret
WRITE_HEX_WORD ENDP

WRITE PROC
        push AX
        mov AH, 9h
        int 21h
        pop AX
        ret
WRITE ENDP

CODE ENDS
END MAIN

```