

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

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

Студент гр. 8383

Бессуднов Г. И.

Преподаватель

Ефремов М. А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры.

Ход работы.

Был написан программный модуль типа .EXE, который выполняет следующие функции:

- 1) Освобождает память для загрузки оверлеев.
- 2) Читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки.
- 3) Файл оверлейного сегмента загружается и выполняется.
- 4) Освобождается память, отведенная для оверлейного сегмента.
- 5) Затем действия 1)-4) выполняются для следующего оверлейного сегмента.

Были написаны и отлажены оверлейные сегменты, которые выводят адрес сегмента, в который они загружены. Исходный код модуля LR_7.EXE представлен в приложении А. Исходный код оверлейных модулей OVL_1.EXE и OVL_2.EXE представлен в приложении Б и В соответственно.

Программа была запущена, когда она находилась в одном каталоге с оверлейными модулями. Результат работы программы представлен на рис. 1.



```
C:\>LR_7.EXE
Free memory
Allocation good
LOAD good
-----
OVL1 adress:  0205
-----
Allocation good
LOAD good
*****
OVL2 adress:  0205
*****
C:\>_
```

Рисунок 1 – Результат выполнения программы

Результат запуска программы в другой директории представлен на рис. 2.

```

Z:\>C:

C:\>cd NEWFOL~1

C:\NEWFOL~1>LR_7.EXE
Free memory
Allocation good
LOAD good

-----
OUL1 adress:  0205
-----

Allocation good
LOAD good
*****
OUL2 adress:  0205
*****

C:\NEWFOL~1>

```

Рисунок 2 – Результат выполнения программы в другой директории

Далее программа была запущена, когда один или оба оверлейных модуля не присутствовали в каталоге. Результаты представлены на рис. 3, 4 и 5 соответственно.

```

C:\NEWFOL~1>LR_7.EXE
Free memory
Allocation good
LOAD good

-----
OUL1 adress:  0205
-----
ALLOCATION ERROR 2: file not found

C:\NEWFOL~1>

```

Рисунок 3 – Результат выполнения программы только с 1-ым оверлеем

```

C:\NEWFOL~1>LR_7.EXE
Free memory
ALLOCATION ERROR 2: file not found
Allocation good
LOAD good
*****
OUL2 adress:  0205
*****

C:\NEWFOL~1>

```

Рисунок 4 – Результат выполнения программы только со 2-ым оверлеем

```

C:\NEWFOL~1>LR_7.EXE
Free memory
ALLOCATION ERROR 2: file not found
ALLOCATION ERROR 2: file not found

C:\NEWFOL~1>

```

Рисунок 5 – Результат выполнения программы без оверлеев

Контрольные вопросы.

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

Так как в .COM модулях содержится PSP, то его необходимо учитывать и делать смещение при обращении на 100h.

Выводы.

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

ПРИЛОЖЕНИЕ А

СОДЕРЖИМОЕ LR_7.ASM

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:ASTACK

ASTACK SEGMENT STACK

DW 200 DUP(?)

ASTACK ENDS

DATA SEGMENT

```
FILE_NAME_1 db "OVL_1.EXE", 0
FILE_NAME_2 db "OVL_2.EXE", 0
OFFSET_PROG dw 0
PATH db 128 dup(0)
MES_ERR_7 db "ERROR 7: destriyed MCB",10,13,"$"
MES_ERR_8 db "ERROR 8: low memory",10,13,"$"
MES_ERR_9 db "ERROR 9: wrong address",10,13,"$"
MES_GOOD db "Free memory",10,13,"$"

MES_ALLOC_GOOD db "Allocation good",10,13,"$"
MES_ALLOC_ERR_2 db "ALLOCATION ERROR 2: file not
found",10,13,"$"
MES_ALLOC_ERR_3 db "ALLOCATION ERROR 3: route not
found",10,13,"$"

MES_LOAD_GOOD db "LOAD good",10,13,"$"
MES_LOAD_ERR1 db "LOAD_MODULE ERROR 1: function not
exist",10,13,"$"
MES_LOAD_ERR2 db "LOAD_MODULE ERROR 2: file not found",10,13,"$"
MES_LOAD_ERR3 db "LOAD_MODULE ERROR 3: route not found",10,13,"$"
MES_LOAD_ERR4 db "LOAD_MODULE ERROR 4: too many opened
files",10,13,"$"
MES_LOAD_ERR5 db "LOAD_MODULE ERROR 5: no accsess",10,13,"$"
MES_LOAD_ERR8 db "LOAD_MODULE ERROR 8: low memory",10,13,"$"
MES_LOAD_ERR10 db "LOAD_MODULE ERROR 10: wrong environment
string",10,13,"$"

DTA_MEM db 43 dup(?)
OVL_ADDRESS dd 0

KEEP_PSP dw 0

DATA_END db 0
DATA ENDS

WRITE_STR PROC NEAR
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret
WRITE_STR ENDP

FREE_MEMORY PROC NEAR
```

```

        push BX
        push DX
        mov BX, offset PROGRAMM_END
        mov AX, offset DATA_END

        add BX, AX
        push CX
        mov CL, 4
        shr BX, CL
        add BX, 2Bh
        pop CX
        mov AH, 4Ah
        int 21h

        jnc MEM_GOOD

        cmp AX, 7
        je ERR_7
        cmp AX, 8
        je ERR_8
        cmp AX, 9
        je ERR_9

ERR_7:
        mov DX, offset MES_ERR_7
        jmp FAIL

ERR_8:
        mov DX, offset MES_ERR_8
        jmp FAIL

ERR_9:
        mov DX, offset MES_ERR_9
        jmp FAIL

MEM_GOOD:
        mov AX, 1
        mov DX, offset MES_GOOD
        call WRITE_STR
        jmp FREE_MEMORY_END

FAIL:
        mov AX, 0
        call WRITE_STR

FREE_MEMORY_END:
        pop DX
        pop BX
        ret
FREE_MEMORY ENDP

ALLOCATE_MEMORY PROC
        push BX
        push CX
        push DX

        push DX

        mov DX, offset DTA_MEM
        mov AH, 1Ah

```

```

        int 21h

        pop DX
        mov CX, 0
        mov AH, 4Eh
        int 21h

        jnc ALLOC_GOOD

        cmp AX, 2
        jmp ALLOC_ERR_2
        cmp AX, 3
        jmp ALLOC_ERR_3

ALLOC_ERR_2:
        mov DX, offset MES_ALLOC_ERR_2
        jmp ALLOC_FAIL
ALLOC_ERR_3:
        mov DX, offset MES_ALLOC_ERR_3
        jmp ALLOC_FAIL

ALLOC_GOOD:
        push DI
        mov DI, offset DTA_MEM
        mov BX, [DI+1Ah]
        mov AX, [DI+1Ch]
        pop DI
        push CX
        mov CL, 4
        shr BX, CL
        mov CL, 12
        shl AX, CL
        pop CX
        add BX, AX
        add BX, 1
        mov AH, 48h
        int 21h
        mov WORD PTR OVL_ADDRESS, AX
        mov DX, offset MES_ALLOC_GOOD
        call WRITE_STR
        mov AX, 1
        jmp ALLOC_END

ALLOC_FAIL:
        mov AX, 0
        call WRITE_STR

ALLOC_END:
        pop DX
        pop CX
        pop BX

        ret
ALLOCATE_MEMORY ENDP

```

```

LOAD_MODULE PROC NEAR

```

```

    push AX
    push BX
    push CX
    push DX
    push DS
    push ES

    mov AX, DATA
    mov ES, AX
    mov DX, offset PATH
    mov BX, offset OVL_ADDRESS
    mov AX, 4B03h
    int 21h

    jnc LOAD_GOOD

    cmp AX, 1
    je LOAD_ERR_1
    cmp AX, 2
    je LOAD_ERR_2
    cmp AX, 3
    je LOAD_ERR_3
    cmp AX, 4
    je LOAD_ERR_4
    cmp AX, 5
    je LOAD_ERR_5
    cmp AX, 8
    je LOAD_ERR_8
    cmp AX, 10
    je LOAD_ERR_10

LOAD_ERR_1:
    mov DX, offset MES_LOAD_ERR1
    jmp LOAD_FAIL
LOAD_ERR_2:
    mov DX, offset MES_LOAD_ERR2
    jmp LOAD_FAIL
LOAD_ERR_3:
    mov DX, offset MES_LOAD_ERR3
    jmp LOAD_FAIL
LOAD_ERR_4:
    mov DX, offset MES_LOAD_ERR4
    jmp LOAD_FAIL
LOAD_ERR_5:
    mov DX, offset MES_LOAD_ERR5
    jmp LOAD_FAIL
LOAD_ERR_8:
    mov DX, offset MES_LOAD_ERR8
    jmp LOAD_FAIL
LOAD_ERR_10:
    mov DX, offset MES_LOAD_ERR10
    jmp LOAD_FAIL

LOAD_GOOD:
    mov DX, offset MES_LOAD_GOOD

```



```

        call WRITE_STR

        mov AX, WORD PTR OVL_ADDRESS
        mov ES, AX
        mov WORD PTR OVL_ADDRESS, 0
        mov WORD PTR OVL_ADDRESS + 2, AX

        call OVL_ADDRESS
        mov ES, AX
        mov AH, 49h
        int 21h
        jmp LOAD_END

LOAD_FAIL:
        call WRITE_STR

LOAD_END:
        pop ES
        pop DS
        pop DX
        pop CX
        pop BX
        pop AX
        ret
LOAD_MODULE ENDP

OVL_EXEC PROC
        push DX
        ;-----

        mov OFFSET_PROG, DX

        mov AX, KEEP_PSP
        mov ES, AX
        mov ES, ES:[2Ch]
        mov BX, 0

PRINT_ENV_VAR:
        cmp BYTE PTR ES:[BX], 0
        je ENV_VAR_END
        inc BX
        jmp PRINT_ENV_VAR
ENV_VAR_END:
        inc BX
        cmp BYTE PTR ES:[BX+1], 0
        jne PRINT_ENV_VAR

        add BX, 2
        mov DI, 0
MARK:
        mov DL, ES:[BX]
        mov BYTE PTR [PATH+DI], DL
        inc BX
        inc DI
        cmp DL, 0

```

```

        je LOOP_END
        cmp DL, '\''
        jne MARK
        mov CX, DI
        jmp MARK
LOOP_END:
        mov DI, CX
        mov SI, OFFSET_PROG
FILE_NAME_LOOP:
        mov DL, BYTE PTR [SI]
        mov BYTE PTR [PATH+DI], DL
        inc DI
        inc SI
        cmp DL, 0
        jne FILE_NAME_LOOP
        ;-----

        mov DX, offset PATH
        call ALLOCATE_MEMORY
        cmp AX, 1
        jne OVL_EXEC_END

        call LOAD_MODULE

OVL_EXEC_END:
        pop DX
        ret
OVL_EXEC ENDP

MAIN PROC
        PUSH DS
        SUB AX, AX
        PUSH AX
        MOV AX, DATA
        MOV DS, AX
        mov KEEP_PSP, ES

        call FREE_MEMORY
        cmp AX, 1
        jne MAIN_END

        mov DX, offset FILE_NAME_1
        call OVL_EXEC

        mov DX, offset FILE_NAME_2
        call OVL_EXEC

MAIN_END:
        xor AL, AL
        mov AH, 4Ch
        int 21h
MAIN ENDP
PROGRAMM_END:

CODE ENDS

END MAIN

```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД OVL_1.ASM

```
CODE SEGMENT
    ASSUME CS:CODE

MAIN PROC FAR
    push AX
    push DX
    push DS
    push DI

    mov AX, CS
    mov DS, AX
    mov DI, offset ADDRESS
    add DI, 18
    call WRD_TO_HEX
    mov DX, offset SEPARATOR
    call WRITE_STR
    MOV DX, offset ADDRESS
    call WRITE_STR
    mov DX, offset SEPARATOR
    call WRITE_STR

    pop DI
    pop DS
    pop DX
    pop AX
    retf
MAIN ENDP

ADDRESS db "OVL1 adress:          ", 13, 10, "$"
SEPARATOR db "-----", 13, 10, "$"

WRITE_STR PROC NEAR
    push AX

    mov AH, 09h
    int 21h

    pop AX
    ret
WRITE_STR 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_TO_HEX PROC NEAR
```

```

        push cx
        mov  ah, al
        call TETR_TO_HEX
        xchg al, ah
        mov  cl, 4
        shr  al, cl
        call TETR_TO_HEX
        pop  cx
        ret
BYTE_TO_HEX ENDP

WRD_TO_HEX PROC NEAR
        push bx
        mov     bh, ah
        call    BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        dec     di
        mov     al, bh
        xor     ah, ah
        call    BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        pop     bx
        ret
WRD_TO_HEX ENDP

CODE ENDS
END MAIN

```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД OVL_2.ASM

```
CODE SEGMENT
    ASSUME CS:CODE

MAIN PROC FAR
    push AX
    push DX
    push DS
    push DI

    mov AX, CS
    mov DS, AX
    mov DI, offset ADDRESS
    add DI, 18
    call WRD_TO_HEX
    mov DX, offset SEPARATOR
    call WRITE_STR
    MOV DX, offset ADDRESS
    call WRITE_STR
    mov DX, offset SEPARATOR
    call WRITE_STR

    pop DI
    pop DS
    pop DX
    pop AX
    retf
MAIN ENDP

ADDRESS db "OVL2 adress:          ", 13, 10, "$"
SEPARATOR db "*****", 13, 10, "$"

WRITE_STR PROC NEAR
    push AX

    mov AH, 09h
    int 21h

    pop AX
    ret
WRITE_STR 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_TO_HEX PROC NEAR
```

```

        push cx
        mov  ah, al
        call TETR_TO_HEX
        xchg al, ah
        mov  cl, 4
        shr  al, cl
        call TETR_TO_HEX
        pop  cx
        ret
BYTE_TO_HEX ENDP

WRD_TO_HEX PROC NEAR
        push bx
        mov     bh, ah
        call    BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        dec     di
        mov     al, bh
        xor     ah, ah
        call    BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        pop     bx
        ret
WRD_TO_HEX ENDP

CODE ENDS
END MAIN

```