

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

Шишкин И.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры.

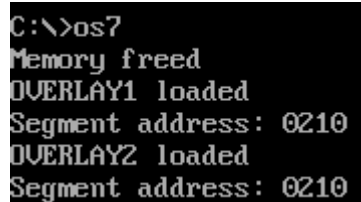
Ход работы.

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

- 1) Освобождает память для загрузки оверлея.
- 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 оверлейные модули OVERLAY1 и OVERLAY2, которые выводят свой сегментный адрес.

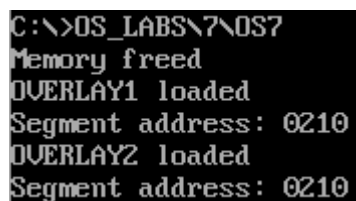
Была запущена отлаженная программа. Результат выполнения программы приведен на рис. 1.



```
C:\>os7
Memory freed
OVERLAY1 loaded
Segment address: 0210
OVERLAY2 loaded
Segment address: 0210
```

Рисунок 1 – Результат работы программы в 1-м случае

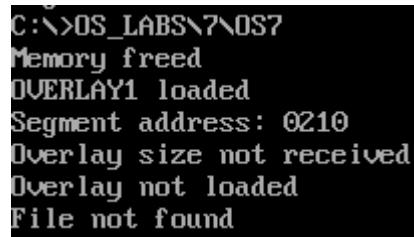
Затем, программа была запущена из другого каталога. Результат выполнения программы приведен на рис. 2.



```
C:\>OS_LABS\7\OS7
Memory freed
OVERLAY1 loaded
Segment address: 0210
OVERLAY2 loaded
Segment address: 0210
```

Рисунок 2 – Результат работы программы во 2-м случае

Была запущена отлаженная программа, когда второго оверлея нет в каталоге. Результат выполнения программы приведен на рис. 3.



```
C:\>OS_LABS\7\OS7
Memory freed
OVERLAY1 loaded
Segment address: 0210
Overlay size not received
Overlay not loaded
File not found
```

Рисунок 3 – Результат работы программы в 3-м случае

Контрольные вопросы.

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

Нужно будет, чтобы обращение к модулю происходило со смещением 100h.

Выводы.

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

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ OS7

Astack SEGMENT STACK

dw 128 dup(0)

Astack ENDS

DATA SEGMENT

PARAMETER_BLOCK dw 0 ;сегментный адрес среды

dd 0 ;сегмент и смещение командной строки

dd 0 ;сегмент и смещение FCB

dd 0 ;сегмент и смещение FCB

STR_OVERLAY1_NAME db 'OVERLAY1.OVL\$'

STR_OVERLAY2_NAME db 'OVERLAY2.OVL\$'

PROGRAM_PATH db 50 dup(0)

OFFSET_OVL_NAME dw 0

DTA_BUFF db 43 dup(0)

OVL_PARAM_SEG dw 0

OVL_ADDRESS dd 0

IS_MEMORY_FREED db 0

STR_FUNCTION_COMPLETED db 'Memory freed', 13, 10, '\$'

STR_FUNC_NOT_COMPLETED db 13, 10, 'Memory is not freed\$'

ERROR_CODE_7 db 13, 10, 'Memory control block destroyed\$'

ERROR_CODE_8 db 13, 10, 'Not enough memory to execute function\$'

ERROR_CODE_9 db 13, 10, 'Invalid memory block address\$'

GET_SIZE_ERROR db 13, 10, 'Overlay size not received\$'

GET_SIZE_ERROR_CODE_2 db 13, 10, 'File not found\$'

GET_SIZE_ERROR_CODE_3 db 13, 10, 'Path not found\$'

LOADING_ERROR db 13, 10, 'Overlay not loaded\$'

LOADING_ERROR_CODE_1 db 13, 10, 'Non-existent function\$'

LOADING_ERROR_CODE_2 db 13, 10, 'File not found\$'

LOADING_ERROR_CODE_3 db 13, 10, 'Path not found\$'

LOADING_ERROR_CODE_4 db 13, 10, 'Too many open files\$'

LOADING_ERROR_CODE_5 db 13, 10, 'No access\$'

LOADING_ERROR_CODE_8 db 13, 10, 'No memory\$'

LOADING_ERROR_CODE_10 db 13, 10, 'Wrong environment\$'

END_OF_DATAA db 0

DATA ENDS

CODE SEGMENT

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

;-----

PRINT PROC near

push AX

mov AH, 09h

int 21h

pop AX

ret

PRINT ENDP

;-----

OVERLAY_LOADING PROC near

call MAKE_FULL_FILE_NAME

call GET_OVERLAY_SIZE

call OVERLAY

ret

OVERLAY_LOADING ENDP

;-----

MAKE_FULL_FILE_NAME PROC near

push AX

push DI

push SI

push ES

;mov OFFSET_OVL_NAME, AX

;mov BX, AX

mov ES, ES:[2Ch] ;смещение до сегмента окружения (environment)

xor DI, DI

NEXT: ;ищем 2 нуля - т.к. строка запуска программы за ними

mov AL, ES:[DI]

;inc DI

cmp AL, 0

je AFTER_FIRST_0

inc DI

jmp NEXT

AFTER_FIRST_0:

inc DI

mov AL, ES:[DI]

cmp AL, 0

```

jne NEXT
add DI, 3h ;нашли 2 нуля, пропускаем 3 цифры
mov SI, 0

```

```

WRITE_NUM:
    mov AL, ES:[DI]
    cmp AL, 0
    je DELETE_FILE_NAME
    mov PROGRAM_PATH[SI], AL
    inc DI
    inc SI
    jmp WRITE_NUM

```

```

DELETE_FILE_NAME:
    dec si
    cmp PROGRAM_PATH[SI], '\'
    je READY
    jmp DELETE_FILE_NAME

```

```

READY:
    mov DI, -1

```

```

ADD_FILE_NAME:
    inc SI
    inc DI
    ;mov AL, OFFSET_OVL_NAME[DI]
    mov AL, BX[DI]
    cmp AL, '$'
    je END_OF_COMMAND_LINE
    mov PROGRAM_PATH[SI], AL
    jmp ADD_FILE_NAME

```

```

END_OF_COMMAND_LINE:
    pop ES
    pop SI
    pop DI
    pop AX
    ret

```

```

MAKE_FULL_FILE_NAME ENDP

```

```

;-----

```

```

GET_OVERLAY_SIZE PROC near
    push AX
    push BX
    push CX

```

```

push DX
push SI

mov AH, 1Ah
mov DX, offset DTA_BUFF
int 21h
; определение размера требуемой памяти
mov AH, 4Eh
mov DX, offset PROGRAM_PATH
mov CX, 0
int 21h
jnc NO_ERRORS

mov DX, offset GET_SIZE_ERROR
call PRINT
mov DX, offset GET_SIZE_ERROR_CODE_2
cmp AX, 2
je WRITE_TYPE_OF_ERROR
mov DX, offset GET_SIZE_ERROR_CODE_3
cmp AX, 3
je WRITE_TYPE_OF_ERROR
jmp END_OF_OVL_SIZE

WRITE_TYPE_OF_ERROR:
    call PRINT
    jmp END_OF_OVL_SIZE

NO_ERRORS:
    mov SI, offset DTA_BUFF
    add SI, 1Ah
    mov BX, [SI]
    shr BX, 4
    mov AX, [SI + 2]
    shl AX, 12
    add BX, AX
    add BX, 2
    mov AH, 48h
    int 21h

    jnc SAVE_SEG
    mov DX, offset STR_FUNC_NOT_COMPLETED
    call PRINT
    jmp END_OF_OVL_SIZE

```

```
SAVE_SEG:
    mov OVL_PARAM_SEG, AX
```

```
END_OF_OVL_SIZE:
    pop SI
    pop DX
    pop CX
    pop BX
    pop AX
    ret
```

```
GET_OVERLAY_SIZE ENDP
```

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

```
OVERLAY PROC NEAR
```

```
    push AX
    push BX
    push DX
    push ES
```

```
    mov DX, offset PROGRAM_PATH
    push DS
    pop ES
    mov BX, offset OVL_PARAM_SEG
    mov AX, 4B03h
```

```
    int 21h
```

```
    jnc NO_LOADING_ERRORS
    mov DX, offset LOADING_ERROR
    call PRINT
    mov DX, offset LOADING_ERROR_CODE_1
    cmp AX, 1
    je WRITE_LOADING_OVL_ERROR
    mov DX, offset LOADING_ERROR_CODE_2
    cmp AX, 2
    je WRITE_LOADING_OVL_ERROR
    mov DX, offset LOADING_ERROR_CODE_3
    cmp AX, 3
    je WRITE_LOADING_OVL_ERROR
    mov DX, offset LOADING_ERROR_CODE_4
    cmp AX, 4
    je WRITE_LOADING_OVL_ERROR
    mov DX, offset LOADING_ERROR_CODE_5
    cmp AX, 5
    je WRITE_LOADING_OVL_ERROR
    mov DX, offset LOADING_ERROR_CODE_8
```



```

cmp AX, 8
je WRITE_LOADING_OVL_ERROR
mov DX, offset LOADING_ERROR_CODE_10
cmp AX, 10
je WRITE_LOADING_OVL_ERROR
jmp END_OF_OVERLAY

```

```

WRITE_LOADING_OVL_ERROR:
    call PRINT
    jmp END_OF_OVERLAY

```

```

NO_LOADING_ERRORS:
    mov AX, OVL_PARAM_SEG
    mov ES, AX
    mov WORD PTR OVL_ADDRESS + 2, AX
    call OVL_ADDRESS
    mov AH, 49h
    int 21h

```

```

END_OF_OVERLAY:
    pop ES
    pop DX
    pop BX
    pop AX
    ret

```

OVERLAY ENDP

```

;-----
FREEING_UP_MEMORY PROC near

```

```

    push AX
    push BX
    push CX
    push DX

```

```

    mov BX, offset END_OF_PROGRAM
    mov AX, offset END_OF_DATA
    add BX, AX
    add BX, 40Fh
    mov CL, 4
    shr BX, CL
    mov AX, 4A00h    ;сжать или расширить блок памяти
    int 21h
    jnc FUNCTION_COMPLETED

```

```

    mov DX, offset STR_FUNC_NOT_COMPLETED

```

```

call PRINT
mov IS_MEMORY_FREED, 0
cmp AX, 7
je IF_ERROR_CODE_7
cmp AX, 8
je IF_ERROR_CODE_8
cmp AX, 9
je IF_ERROR_CODE_9

IF_ERROR_CODE_7:
    mov DX, offset ERROR_CODE_7
    call PRINT
    jmp END_OF_FREEING
IF_ERROR_CODE_8:
    mov DX, offset ERROR_CODE_8
    call PRINT
    jmp END_OF_FREEING
IF_ERROR_CODE_9:
    mov DX, offset ERROR_CODE_9
    call PRINT
    jmp END_OF_FREEING

FUNCTION_COMPLETED:
    mov DX, offset STR_FUNCTION_COMPLETED
    call PRINT
    mov IS_MEMORY_FREED, 1

END_OF_FREEING:
    pop DX
    pop CX
    pop BX
    pop AX
    ret
FREEING_UP_MEMORY ENDP
;-----
BEGIN PROC FAR
    xor AX, AX
    push AX
    mov AX, DATA
    mov DS, AX
    mov BX, DS

    call FREEING_UP_MEMORY
    cmp IS_MEMORY_FREED, 1

```

```
jne ENDD
; 1-я оверлейная программа
mov BX, offset STR_OVERLAY1_NAME
call OVERLAY_LOADING
; 2-я оверлейная программа
mov BX, offset STR_OVERLAY2_NAME
call OVERLAY_LOADING

ENDD:
    xor AL, AL
    mov AH, 4Ch
    int 21h
BEGIN ENDP
END_OF_PROGRAM:
CODE ENDS
END BEGIN
```

ПРИЛОЖЕНИЕ Б
КОД ПРОГРАММЫ OVERLAY1

CODE SEGMENT

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

;-----

BEGIN PROC FAR

push AX

push DX

push DS

push DI

mov AX, CS

mov DS, AX

mov DX, offset IF_LOAD

call PRINT

mov DI, offset SEGMENT_ADDRESS

add DI, 22

call WRD_TO_HEX

mov DX, offset SEGMENT_ADDRESS

call PRINT

pop DI

pop DS

pop DX

pop AX

RETF

BEGIN ENDP

;-----

PRINT PROC near

push AX

mov AH, 09h

int 21h

pop AX

ret

PRINT 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
;байт в AL переводится в два символа шест. числа в AX
    push CX
    mov AH, AL
    call TETR_TO_HEX
    xchg AL, AH
    mov CL, 4
    shr AL, CL
    call TETR_TO_HEX ;в AL старшая цифра
    pop CX ;в AH младшая
    ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
    push BX
    mov BH, AH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    dec DI
    mov AL, BH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    pop BX
    ret
WRD_TO_HEX ENDP
;-----
    IF_LOAD db 13, 10, 'OVERLAY1 loaded$'
    SEGMENT_ADDRESS db 13, 10, 'Segment address:    $'
;-----
CODE ENDS

```

END BEGIN

ПРИЛОЖЕНИЕ В

КОД ПРОГРАММЫ OVERLAY2

CODE SEGMENT

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

;-----

BEGIN PROC FAR

push AX

push DX

push DS

push DI

mov AX, CS

mov DS, AX

mov DX, offset IF_LOAD

call PRINT

mov DI, offset SEGMENT_ADDRESS

add DI, 22

call WRD_TO_HEX

mov DX, offset SEGMENT_ADDRESS

call PRINT

pop DI

pop DS

pop DX

pop AX

RETF

BEGIN ENDP

;-----

PRINT PROC near

push AX

mov AH, 09h

int 21h

pop AX

ret

PRINT 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
;байт в AL переводится в два символа шест. числа в AX
    push CX
    mov AH, AL
    call TETR_TO_HEX
    xchg AL, AH
    mov CL, 4
    shr AL, CL
    call TETR_TO_HEX ;в AL старшая цифра
    pop CX ;в AH младшая
    ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
    push BX
    mov BH, AH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    dec DI
    mov AL, BH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    pop BX
    ret
WRD_TO_HEX ENDP
;-----
    IF_LOAD db 13, 10, 'OVERLAY2 loaded$'
    SEGMENT_ADDRESS db 13, 10, 'Segment address:    $'
;-----
CODE ENDS

```


END BEGIN