

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. 8382

Колногоров Д.Г.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры.

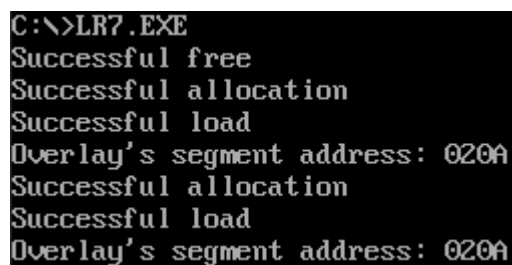
Выполнение работы.

Был написан программный модуль типа **.EXE** (представлен в приложении А), 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.

Также были написаны оверлейные сегменты (код представлен в приложении Б). Оверлейный сегмент выводит адрес сегмента, в который он загружен.

На рисунке 1 представлен результат работы программы.



```
C:\>LR7.EXE
Successful free
Successful allocation
Successful load
Overlay's segment address: 020A
Successful allocation
Successful load
Overlay's segment address: 020A
```

Рисунок 1 — результат работы программы

На рисунке 2 представлен результат запуска программы из каталога TEST.

```
C:\TEST>LR7.EXE
Successful free
Successful allocation
Successful load
Overlay's segment address: 020A
Successful allocation
Successful load
Overlay's segment address: 020A
```

Рисунок 2 — результат запуска программы из каталога TEST

На рисунке 3 представлен результат работы программы при отсутствии в каталоге первого запускаемого оверлея.

```
C:\TEST>LR7.EXE
Successful free
SIZE ERROR 2: file not found
Successful allocation
Successful load
Overlay's segment address: 020A
```

Рисунок 3 — результат работы программы при отсутствии первого оверлея в каталоге

Контрольные вопросы.

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

Программа корректно работает в случае, если в качестве оверлейного сегмента использовать .COM модуль. Необходимо лишь учитывать смещение в 100h (PSP) для .COM модуля.

Вывод.

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

ПРИЛОЖЕНИЕ А

СОДЕРЖИМОЕ ФАЙЛА LR7.ASM

DATA SEGMENT

NEW_LINE db 10,13,'\$'

PROGRAM_TO_RUN1 db "OVERLAY1.EXE",0

PROGRAM_TO_RUN2 db "OVERLAY2.EXE",0

PROGRAM_TO_RUN_OFFSET dw 0

PROGRAM_TO_RUN_FULL_PATH db 128 dup(0)

STR_MEMORY_SUCCESS db "Successful free",10,13,"\$"

STR_MEMORY_ERROR7 db "FREE ERROR 7: controlling block is
destroyed",10,13,"\$"

STR_MEMORY_ERROR8 db "FREE ERROR 8: not enough memory for
function",10,13,"\$"

STR_MEMORY_ERROR9 db "FREE ERROR 9: wrong memory block address",10,13,"\$"

STR_SIZE_SUCCESS db "Successful allocation",10,13,"\$"

STR_SIZE_ERROR2 db "SIZE ERROR 2: file not found",10,13,"\$"

STR_SIZE_ERROR3 db "SIZE ERROR 3: route not found",10,13,"\$"

STR_LOAD_SUCCESS db "Successful load",10,13,"\$"

STR_LOAD_ERROR1 db "SIZE ERROR 1: wrong function number",10,13,"\$"

STR_LOAD_ERROR2 db "SIZE ERROR 2: file not found",10,13,"\$"

STR_LOAD_ERROR3 db "SIZE ERROR 3: route not found",10,13,"\$"

STR_LOAD_ERROR4 db "SIZE ERROR 4: too many open files",10,13,"\$"

STR_LOAD_ERROR5 db "SIZE ERROR 5: access denied",10,13,"\$"

STR_LOAD_ERROR8 db "SIZE ERROR 8: not enough memory",10,13,"\$"

STR_LOAD_ERROR10 db "SIZE ERROR 10: wrong environment",10,13,"\$"

MEMORY_FOR_DTA db 43 dup(?)

OVERLAY_SEG_ADDRESS dd 0

KEEP_PSP dw 0

END_OF_DATA db 0

DATA ENDS

```
AStack SEGMENT STACK
```

```
    DW 200 DUP(?)
```

```
AStack ENDS
```

```
CODE SEGMENT
```

```
    ASSUME  CS:CODE, DS:DATA, ES:NOTHING, SS:AStack
```

```
PRINT_NEW_LINE PROC NEAR
```

```
    push DX
```

```
    push AX
```

```
    mov DX, offset NEW_LINE
```

```
    mov AH, 09h
```

```
    int 21h
```

```
    pop AX
```

```
    pop DX
```

```
    ret
```

```
PRINT_NEW_LINE ENDP
```

```
PRINT_STRING PROC
```

```
    push DX
```

```
    push AX
```

```
    mov AH, 09h
```

```
    int 21h
```

```
    pop AX
```

```
    pop DX
```

```
    ret
```

```
PRINT_STRING ENDP
```

```
PREPARE_MEMORY PROC
```

```
    ; AX=1 if successful free
```

```
    ; otherwise AX=0
```

```
    push BX
```

```
    push DX
```

```
    mov BX, offset END_OF_PROGRAM
```

```
    mov AX, offset END_OF_DATA
```

```

add BX, AX

shr BX, 1
shr BX, 1
shr BX, 1
shr BX, 1
add BX, 2Bh

mov AH, 4Ah
int 21h

jnc MEMORY_SUCCESS

cmp AX, 7
jmp MEMORY_ERROR7
cmp AX, 8
jmp MEMORY_ERROR7
cmp AX, 9
jmp MEMORY_ERROR7

MEMORY_ERROR7:
    mov DX, offset STR_MEMORY_ERROR7
    jmp MEMORY_FAIL
MEMORY_ERROR8:
    mov DX, offset STR_MEMORY_ERROR8
    jmp MEMORY_FAIL
MEMORY_ERROR9:
    mov DX, offset STR_MEMORY_ERROR9
    jmp MEMORY_FAIL

MEMORY_SUCCESS:
    mov AX, 1
    mov DX, offset STR_MEMORY_SUCCESS
    call PRINT_STRING
    jmp PREPARE_MEMORY_END

MEMORY_FAIL:
    mov AX, 0
    call PRINT_STRING

PREPARE_MEMORY_END:

```

```

        pop DX
        pop BX

        ret
PREPARE_MEMORY ENDP

PREPARE_PATH PROC
        ; DX=offset of program's name
        push AX
        push CX
        push BX
        push DI
        push SI
        push ES

        mov PROGRAM_TO_RUN_OFFSET, DX

        ; set ES to env variables segment
        mov AX, KEEP_PSP
        mov ES, AX
        mov ES, ES:[2Ch]

        mov BX, 0
SKIP_ENV_VARIABLE:
        cmp BYTE PTR ES:[BX], 0
        je VARIABLE_END
        inc BX
        jmp SKIP_ENV_VARIABLE
VARIABLE_END:
        inc BX
        cmp BYTE PTR ES:[BX+1], 0
        jne SKIP_ENV_VARIABLE

        add BX, 2    ; skip 0 and space

        mov DI, 0
PATH_LOOP:
        mov DL, ES:[BX]
        mov BYTE PTR [PROGRAM_TO_RUN_FULL_PATH+DI], DL
        inc BX
        inc DI

```

```

        cmp DL, 0
        je PATH_LOOP_END
        cmp DL, '\'
        jne PATH_LOOP
        mov CX, DI
        jmp PATH_LOOP
PATH_LOOP_END:
mov DI, CX

mov SI, PROGRAM_TO_RUN_OFFSET
FILENAME_LOOP:
        mov DL, BYTE PTR [SI]
        mov BYTE PTR [PROGRAM_TO_RUN_FULL_PATH+DI], DL
        inc DI
        inc SI
        cmp DL, 0
        jne FILENAME_LOOP

pop ES
pop SI
pop DI
pop BX
pop CX
pop AX

ret
PREPARE_PATH ENDP

ALLOCATE_MEM_FOR_OVERLAY PROC
        ; input: DX=overlay file name
        push BX
        push CX
        push DX

        push DX                ; save file name

        ; set DTA address
        mov DX, offset MEMORY_FOR_DTA
        mov AH, 1Ah
        int 21h

```



```

; get size of overlay
pop DX                ; restore file name
mov CX, 0
mov AH, 4Eh
int 21h

jnc SIZE_SUCCESS

cmp AX, 2
jmp SIZE_ERROR2
cmp AX, 3
jmp SIZE_ERROR3

SIZE_ERROR2:
    mov DX, offset STR_SIZE_ERROR2
    jmp SIZE_FAIL
SIZE_ERROR3:
    mov DX, offset STR_SIZE_ERROR3
    jmp SIZE_FAIL

SIZE_SUCCESS:
    ; get size from DTA
    push DI
    mov DI, offset MEMORY_FOR_DTA
    mov BX, [DI+1Ah]    ; low
    mov AX, [DI+1Ch]    ; high
    pop DI

    ; convert to paragraphs
    push CX
    mov CL, 4
    shr BX, CL
    mov CL, 12
    shl AX, CL
    pop CX
    add BX, AX
    add BX, 1

    ; allocate memory
    mov AH, 48h
    int 21h

```

```

        mov WORD PTR OVERLAY_SEG_ADDRESS, AX

        mov DX, offset STR_SIZE_SUCCESS
        call PRINT_STRING

        mov AX, 1
        jmp SIZE_END

SIZE_FAIL:
        mov AX, 0
        call PRINT_STRING

SIZE_END:

        pop DX
        pop CX
        pop BX

        ret
ALLOCATE_MEM_FOR_OVERLAY ENDP

LOAD_PROGRAM PROC
        push AX
        push BX
        push CX
        push DX
        push DS
        push ES

        mov AX, DATA
        mov ES, AX
        mov DX, offset PROGRAM_TO_RUN_FULL_PATH
        mov BX, offset OVERLAY_SEG_ADDRESS
        mov AX, 4B03h
        int 21h

        jnc LOAD_SUCCESS

        cmp AX, 1
        jmp LOAD_ERROR1

```

```

cmp AX, 2
jmp LOAD_ERROR2
cmp AX, 3
jmp LOAD_ERROR3
cmp AX, 4
jmp LOAD_ERROR4
cmp AX, 5
jmp LOAD_ERROR5
cmp AX, 8
jmp LOAD_ERROR8
cmp AX, 10
jmp LOAD_ERROR10

LOAD_ERROR1:
    mov DX, offset STR_LOAD_ERROR1
    jmp LOAD_FAIL
LOAD_ERROR2:
    mov DX, offset STR_LOAD_ERROR2
    jmp LOAD_FAIL
LOAD_ERROR3:
    mov DX, offset STR_LOAD_ERROR3
    jmp LOAD_FAIL
LOAD_ERROR4:
    mov DX, offset STR_LOAD_ERROR4
    jmp LOAD_FAIL
LOAD_ERROR5:
    mov DX, offset STR_LOAD_ERROR5
    jmp LOAD_FAIL
LOAD_ERROR8:
    mov DX, offset STR_LOAD_ERROR8
    jmp LOAD_FAIL
LOAD_ERROR10:
    mov DX, offset STR_LOAD_ERROR10
    jmp LOAD_FAIL

LOAD_SUCCESS:
    mov DX, offset STR_LOAD_SUCCESS
    call PRINT_STRING

    ; swap seg and offset
    mov AX, WORD PTR OVERLAY_SEG_ADDRESS

```

```

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

        ; call overlay
        call OVERLAY_SEG_ADDRESS

        ; free memory
        mov ES, AX
        mov AH, 49h
        int 21h

        jmp LOAD_END

LOAD_FAIL:
        call PRINT_NEW_LINE
        call PRINT_STRING

LOAD_END:

        pop ES
        pop DS
        pop DX
        pop CX
        pop BX
        pop AX

        ret
LOAD_PROGRAM ENDP

RUN_OVERLAY PROC
        ; DX=offset of overlay to run
        push DX

        call PREPARE_PATH

        mov DX, offset PROGRAM_TO_RUN_FULL_PATH
        call ALLOCATE_MEM_FOR_OVERLAY
        cmp AX, 1
        jne RUN_OVERLAY_END

```

```

        call LOAD_PROGRAM

RUN_OVERLAY_END:
        pop DX
        ret
RUN_OVERLAY ENDP

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

        call PREPARE_MEMORY
        cmp AX, 1
        jne MAIN_END

        mov DX, offset PROGRAM_TO_RUN1
        call RUN_OVERLAY

        mov DX, offset PROGRAM_TO_RUN2
        call RUN_OVERLAY

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

CODE ENDS

END MAIN

```

ПРИЛОЖЕНИЕ Б

СОДЕРЖИМОЕ ФАЙЛА OVERLAY.ASM

CODE SEGMENT

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

MAIN PROC FAR

 jmp MAIN_START

 STR_OVERLAY_ADDRESS db "Overlay's segment address: \$"

 STR_NEW_LINE db 10,13,'\$'

MAIN_START:

 push AX

 push DX

 push DS

 mov AX, CS

 mov DS, AX

 mov DX, offset STR_OVERLAY_ADDRESS

 mov AH, 9h

 int 21h

 mov AX, CS

 call PRINT_WORD

 mov DX, offset STR_NEW_LINE

 mov AH, 9h

 int 21h

 pop DS

 pop DX

 pop AX

 retf

MAIN ENDP

PRINT_WORD PROC

 xchg AH, AL

 call PRINT_BYTE

 xchg AH, AL

 call PRINT_BYTE

```

        ret
PRINT_WORD ENDP

PRINT_BYTE PROC
; prints AL as two hex digits
    push AX
    push BX
    push DX

    call BYTE_TO_HEX
    mov BH, AH

    mov DL, AL
    mov AH, 02h
    int 21h

    mov DL, BH
    mov AH, 02h
    int 21h

    pop DX
    pop BX
    pop AX
    ret
PRINT_BYTE ENDP

TETR_TO_HEX PROC
    and     AL, 0Fh
    cmp     AL, 09
    jbe     NEXT
    add     AL, 07
NEXT:
    add     AL, 30h
    ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC
; AL --> two hex symbols in AX
    push    CX
    mov     AH, AL
    call    TETR_TO_HEX

```

```
        xchg     AL,AH
        mov      CL,4
        shr      AL,CL
        call     TETR_T0_HEX ; AL - high digit
        pop      CX          ; AH - low digit
        ret
BYTE_TO_HEX ENDP
```

CODE ENDS

END MAIN