

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

Студент гр. 8383

Костарев К.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры.

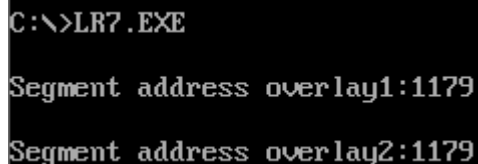
Выполнение работы.

Для выполнения лабораторной работы был написан и отлажен программный модуль типа .EXE, в котором происходит:

- 1) освобождение памяти для загрузки оверлеев;
- 2) чтение размера файла оверлея и выделение памяти, достаточной для его загрузки;
- 3) загрузка и выполнение оверлейного сегмента;
- 4) освобождение памяти, отведённой для оверлейного сегмента;
- 5) повторение пунктов 1-4 для следующего оверлейного сегмента.

Исходный код программы представлен в приложении А.

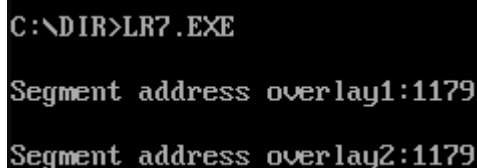
Для начала программа была запущена, когда загрузочный и оверлейные модули находятся в текущем каталоге. На рис. 1 представлен результат работы программы.



```
C:\>LR7.EXE  
  
Segment address overlay1:1179  
Segment address overlay2:1179
```

Рисунок 1 – Случай, когда все модули в текущем каталоге

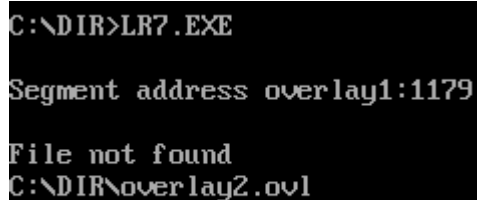
Далее программа была запущена, когда оба модуля находились не в текущем каталоге. Результат работы программы представлен на рис. 2.



```
C:\>DIR>LR7.EXE  
  
Segment address overlay1:1179  
Segment address overlay2:1179
```

Рисунок 2 – Случай, когда оба модуля не в текущем каталоге

Работа программы в том случае, когда первый оверлейный модуль находится не в текущем каталоге, представлена на рис. 3.



```
C:\DIR>LR7.EXE
Segment address overlay1:1179
File not found
C:\DIR\overlay2.ovl
```

Рисунок 3 – Случай, когда первый оверлейный модуль не в текущем каталоге

Наконец, работа программы в том случае, когда второй оверлейный модуль находится не в текущем каталоге, представлена на рис. 4.



```
C:\DIR>LR7.EXE
File not found
C:\DIR\overlay1.ovl
```

Рисунок 4 – Случай, когда второй оверлейный модуль не в текущем каталоге

Программа аварийно завершает работу, не найдя первый оверлейный модуль, как этого требует задание.

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

В начале выделенной памяти поместить PSP и увеличить смещение оверлейного сегмента на 256 байт, так как PSP не будет сформирован при таком запуске.

Выводы.

В ходе выполнения данной лабораторной работы была изучена возможность построения загрузочного модуля оверлейной структуры и реализован интерфейс вызывающего и оверлейных модулей.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ВЫЗЫВАЮЩЕГО МОДУЛЯ

```
AStack      SEGMENT      STACK
            db      512    dup(0)
AStack ENDS

DATA SEGMENT
    ERROR_MEM_7 db 13, 10, 'MCB destroyed', 13, 10, '$'
    ERROR_MEM_8 db 13, 10, 'Not enough memory', 13, 10, '$'
    ERROR_MEM_9 db 13, 10, 'Wrong address', 13, 10, '$'
    ERROR_LOAD_1 db 13, 10, 'Number of function is wrong', 13, 10, '$'
    ERROR_LOAD_2 db 13, 10, 'File not found', 13, 10, '$'
    ERROR_LOAD_3 db 'Rout was not found', 13, 10, '$'
    ERROR_LOAD_4 db 'Too many files open', 13, 10, '$'
    ERROR_LOAD_5 db 13, 10, 'Disk error', 13, 10, '$'
    ERROR_LOAD_8 db 13, 10, 'Insufficient value of memory', 13, 10, '$'
    ERROR_LOAD_10 db 13, 10, 'Incorrect environment string', 13, 10, '$'
    ERROR_ALLOCATE db 'Failed to allocate memory', 13, 10, '$'
    ERROR_NO_LOAD db 'Overlay was not been loaded: '
    PATH db 256 dup(0), '$'
    O_DATA db 43 dup(0)
    KEEP_PSP dw 0
    ADDRESS_CALL dd 0
    ADDRESS_BLOCK dw 0
    OVERLAY_1 db 'overlay1.ovl', 0
    OVERLAY_2 db 'overlay2.ovl', 0
DATA ENDS
DUM_SEGMENT SEGMENT
DUM_SEGMENT ENDS

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

WRITE_STRING PROC near
    push AX
    mov AH, 09H
    int 21H
    pop AX
    ret
WRITE_STRING ENDP

FREE PROC
    mov BX, offset DUM_SEGMENT
    mov AX, ES
    sub BX, AX
    mov CL, 4H
    shr BX, CL
    mov AH, 4AH
```

```

    int 21H
    jnc GOOD
    cmp AX, 7
    mov DX, offset ERROR_MEM_7
    je HAVE_ERROR
    cmp AX, 8
    mov DX, offset ERROR_MEM_8
    je HAVE_ERROR
    cmp AX, 9
    mov DX, offset ERROR_MEM_9
HAVE_ERROR:
    call WRITE_STRING
    xor AL,AL
    mov AH,4CH
    int 21H
GOOD:
    ret
FREE ENDP

```

```

FIND_PATH PROC
    push DS
    push DX
    mov DX, seg O_DATA
    mov DS, DX
    mov DX, offset O_DATA
    mov AH, 1AH
    int 21H
    pop DX
    pop DS
    push ES
    push DX
    push AX
    push BX
    push CX
    push DI
    push SI
    mov ES, KEEP_PSP
    mov AX, ES:[2CH]
    mov ES, AX
    xor BX, BX
FIND_ZERO:
    mov AL, ES:[BX]
    cmp AL, 0H
    je END_FIND_ZERO
    inc BX
    jmp FIND_ZERO
END_FIND_ZERO:
    inc BX
    cmp byte ptr ES:[BX], 0H
    jne FIND_ZERO

```

```

        add BX, 3H
        mov SI, offset PATH
WRITE_PATH:
        mov AL, ES:[BX]
        mov [SI], AL
        inc SI
        cmp AL, 0H
        je END_WRITE_PATH
        inc BX
        jmp WRITE_PATH
END_WRITE_PATH:
        sub SI, 8H
        mov DI, BP
ENTRY_WAY:
        mov AH, [DI]
        mov [SI], AH
        cmp AH, 0H
        je END_FIND_PATH
        inc DI
        inc SI
        jmp ENTRY_WAY
END_FIND_PATH:
        pop SI
        pop DI
        pop CX
        pop BX
        pop AX
        pop DX
        pop ES
        ret
FIND_PATH ENDP

OVERLAY_FND PROC
        push DS
        push     DX
        push     CX

        mov AH, 4EH
        xor CX, CX
        mov DX, offset PATH
        int 21H
        jnc OVERLAY_NO_ERROR
        cmp AX, 3
        je WRITE_ERROR_3
        mov DX, offset ERROR_LOAD_2
        jmp END_ERROR
WRITE_ERROR_3:
        mov     DX, offset ERROR_LOAD_3
END_ERROR:
        call WRITE_STRING

```

```

    mov DX, offset PATH
    call WRITE_STRING
    pop CX
    pop DX
    pop DS
    xor AL, AL
    mov AH, 4CH
    int 21H
OVERLAY_NO_ERROR:
    push ES
    push     BX
    mov BX, offset O_DATA
    mov DX, [BX+1CH]
    mov AX, [BX+1AH]
    mov CL, 4H
    shr AX, CL
    mov CL, 12
    sal DX, CL
    add AX, DX
    inc AX
    mov BX, AX
    mov AH, 48H
    int 21H
    jnc END_SUCSESS
    mov DX, offset ERROR_ALLOCATE
    call WRITE_STRING
    xor AL, AL
    mov AH, 4CH
    int 21H
END_SUCSESS:
    mov ADDRESS_BLOCK, AX
    pop BX
    pop ES
    pop CX
    pop DX
    pop DS
    ret
OVERLAY_FND ENDP

CALL_OVERLAY PROC
    push DX
    push BX
    push AX
    mov BX, seg ADDRESS_BLOCK
    mov ES, BX
    mov BX, offset ADDRESS_BLOCK
    mov DX, seg PATH
    mov DS, DX
    mov DX, offset PATH
    push SS

```

```

push SP
mov AX, 4B03H
int 21H
push DX
jnc NO_ERROR
mov DX, offset ERROR_NO_LOAD
call WRITE_STRING
cmp AX, 1
mov DX, offset ERROR_LOAD_1
je WRITE_ERROR
cmp AX, 2
mov DX, offset ERROR_LOAD_2
je WRITE_ERROR
cmp AX, 3
mov DX, offset ERROR_LOAD_3
je WRITE_ERROR
cmp AX, 4
mov DX, offset ERROR_LOAD_4
je WRITE_ERROR
cmp AX, 5
mov DX, offset ERROR_LOAD_5
je WRITE_ERROR
cmp AX, 8
mov DX, offset ERROR_LOAD_8
je WRITE_ERROR
cmp AX, 10
mov DX, offset ERROR_LOAD_10
WRITE_ERROR:
    call WRITE_STRING
    jmp END_CALL
NO_ERROR:
    mov     AX, DATA
    mov     DS, AX
    mov     AX, ADDRESS_BLOCK
    mov     word ptr ADDRESS_CALL+2, AX
    call ADDRESS_CALL
    mov     AX, ADDRESS_BLOCK
    mov     ES, AX
    mov     AX, 4900H
    int     21h
    mov     AX, DATA
    mov     DS, AX
END_CALL:
    pop     DX
    pop     SP
    pop     SS
    mov     ES, KEEP_PSP
    pop     AX
    pop     BX
    pop     DX

```



```

    ret
CALL_OVERLAY ENDP

MAIN PROC NEAR
    mov AX, DATA
    mov DS, AX
    mov KEEP_PSP, ES
    call FREE
    mov BP, offset OVERLAY_1
    call FIND_PATH
    call OVERLAY_FND
    call CALL_OVERLAY
    sub BX, BX
    mov BP, offset OVERLAY_2
    call FIND_PATH
    call OVERLAY_FND
    call CALL_OVERLAY
    xor AL, AL
    mov AX, 4C00H
    int 21H
MAIN ENDP
CODE ENDS
    END MAIN

```