

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

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

Студент гр. 9383

Хотяков Е.П.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

Цель работы.

Исследование структуры оверлейного сегмента и способа загрузки и выполнения оверлейных сегментов. Написание программы, состоящей из нескольких модулей.

Задание.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет 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.

Шаг 2. Также необходимо написать и отладить оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

Шаг 3. Запустите отлаженное приложение. Оверлейные сегменты должны загружаться с одного и того же адреса, перекрывая друг друга.

Шаг 4. Запустите приложение из другого каталога. Приложение должно быть выполнено успешно.

Шаг 5. Запустите приложение в случае, когда одного оверлея нет в каталоге. Приложение должно закончиться аварийно.

Результаты исследования проблем.

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет, требуемые в задании функции.

Шаг 2. Были написаны и отлажены оверлейные сегменты, они выводят адрес сегмента, в который они загружены.

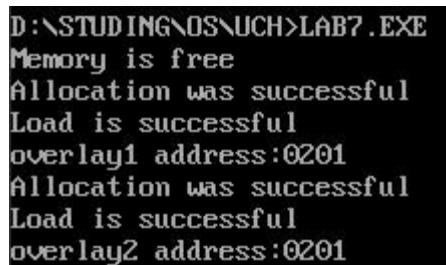
Шаг 3. Программа была запущена для того, чтобы убедиться, что оверлейные сегменты загружаются с одного и того же адреса, перекрывая друг друга.



```
D:\STUDING\OS\UCH\MASM>LAB7.EXE
Memory is free
Allocation was successful
Load is successful
overlay1 address:0201
Allocation was successful
Load is successful
overlay2 address:0201
```

Рисунок 1 – Демонстрация корректной работы программы(ovl1 и ovl2 модули имеют одинаковый адрес).

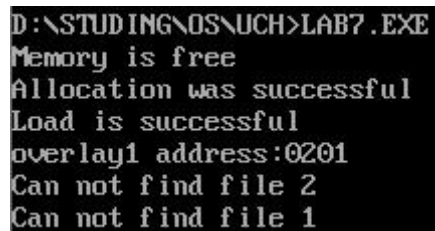
Шаг 4. Программа была запущена из другого каталога, чтобы убедиться в ее работоспособности.



```
D:\STUDING\OS\UCH>LAB7.EXE
Memory is free
Allocation was successful
Load is successful
overlay1 address:0201
Allocation was successful
Load is successful
overlay2 address:0201
```

Рисунок 2 – Демонстрация корректной работы программы при запуске из другого каталога.

Шаг 4. Программа была запущена, после того, как из каталога был перемещен файл ovl2.ovl, для того, чтобы убедиться, что программа корректно обрабатывает ошибки.



```
D:\STUDING\OS\UCH>LAB7.EXE
Memory is free
Allocation was successful
Load is successful
overlay1 address:0201
Can not find file 2
Can not find file 1
```

Рисунок 3 – Демонстрация корректной обработки ошибок программы, если оверлейный файл находится в другом каталоге.

По итогам выполнения работы можно ответить на контрольные вопросы:

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

После того как произойдет запись значений регистров в стек, надо положить значение регистра CS в DS, так как адрес сегмента данных совпадает с сегментом кода. А также, по причине того, что сегменты настроены на PSP, следует добавить 100h.

Выводы.

Исследованы структуры оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Написана программы, состоящая из нескольких модулей.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Lab7.asm:

```
MY_STACK SEGMENT STACK
    DW 64 DUP(?)
MY_STACK ENDS

DATA SEGMENT
    PARAMS_BLOCK DW 0
                    DD 0
                    DD 0
                    DD 0
    NEW_COMMAND_LINE DB 1H, 0DH
    PATH_ DB 128 DUP(0)
    OVERLAY_ADDRESS DD 0
    FILE_OVERLAY1 DB "OVERLAY1.OVL", 0
    FILE_OVERLAY2 DB "OVERLAY2.OVL", 0
    SAVED DW 0
    CUR_OVERLAY DW 0
    SAVED_SP DW 0
    SAVED_SS DW 0
    DTA DB 43 DUP(0); БУФФЕР ДЛЯ DTA
    SAVED_PSP DW 0

    FREE_MEM_SUCCESS DB "MEMORY IS FREE", 0DH, 0AH, '$'
    CONTROL_BLOCK_ERROR DB "CONTROL BLOCK WAS DESTROYED", 0DH, 0AH,
'$'
    FUNCTION_MEM_ERROR DB "NOT ENOUGH MEMORY FOR FUNCTION", 0DH, 0AH,
'$'
    WRONG_ADDRESS DB "WRONG ADDRESS FOR BLOCK OF MEMORY", 0DH, 0AH,
'$'

    WRONG_NUMBER_ERROR DB "WRONG FUNCTION NUMBER", 0DH, 0AH, '$'
    CANT_FIND_ERROR DB "CAN NOT FIND FILE 1", 0DH, 0AH, '$'
    PATH_ERROR DB "CAN NOT FIND PATH 1", 0DH, 0AH, '$'
    OPEN_ERROR DB "TOO MUCH OPPENED FILES", 0DH, 0AH, '$'
    ACCESS_ERROR DB "NO ACCESS FOR FILE", 0DH, 0AH, '$'
    NOT_ENOUGH_MEM_ERROR DB "NOT ENOUGH MEMORY", 0DH, 0AH, '$'
    ENVIRONMENT_ERROR DB "WRONG ENVIRONMENT", 0DH, 0AH, '$'

    CANT_FIND_ERROR2 DB "CAN NOT FIND FILE 2", 0DH, 0AH, '$'
    PATH_ERROR2 DB "CAN NOT FIND PATH 2", 0DH, 0AH, '$'

    NORMAL_END DB "LOAD IS SUCCESSFUL", 0DH, 0AH, '$'
    NORMAL_ALLOC_END DB "ALLOCATION WAS SUCCESSFUL", 0DH, 0AH, '$'

    ERR_FLAG DB 0

    DATA_END DB 0
DATA ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:MY_STACK
```

```

WRITE_STRING PROC NEAR
    PUSH AX
    MOV AH, 9H
    INT 21H
    POP AX
    RET
WRITE_STRING ENDP

FREE_MEM PROC NEAR
    PUSH AX
    PUSH BX
    PUSH DX
    PUSH CX

    MOV AX, OFFSET DATA_END
    MOV BX, OFFSET PROC_END
    ADD BX, AX
    MOV CL, 4
    SHR BX, CL
    ADD BX, 2BH

    MOV AH, 4AH
    INT 21H

    JNC FMS
    MOV ERR_FLAG, 1

    CMP AX, 7
    JE CBE
    CMP AX, 8
    JE FME
    CMP AX, 9
    JE WA

CBE:
    MOV DX, OFFSET CONTROL_BLOCK_ERROR
    CALL WRITE_STRING
    JMP FREE_MEM_END
FME:
    MOV DX, OFFSET FUNCTION_MEM_ERROR
    CALL WRITE_STRING
    JMP FREE_MEM_END
WA:
    MOV DX, OFFSET WRONG_ADDRESS
    CALL WRITE_STRING
    JMP FREE_MEM_END
FMS:
    MOV DX, OFFSET FREE_MEM_SUCCESS
    CALL WRITE_STRING

FREE_MEM_END:
    POP DX
    POP BX
    POP CX
    POP AX
    RET

```

```

FREE_MEM ENDP

PATH PROC NEAR
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH DI
    PUSH SI
    PUSH ES

    MOV SAVED, DX
    MOV AX, SAVED_PSP
    MOV ES, AX
    MOV ES, ES:[2CH]
    MOV BX, 0

FIND_PATH:
    INC BX
    CMP BYTE PTR ES:[BX-1], 0
    JNE FIND_PATH
    CMP BYTE PTR ES:[BX+1], 0
    JNE FIND_PATH
    ADD BX, 2
    MOV DI, 0

FIND_LOOP_1:
    MOV DL, ES:[BX]
    MOV BYTE PTR [PATH_ + DI], DL
    INC DI
    INC BX
    CMP DL, 0
    JE END_FIND_LOOP
    CMP DL, '\'
    JNE FIND_LOOP_1
    MOV CX, DI
    JMP FIND_LOOP_1

END_FIND_LOOP:
    MOV DI, CX
    MOV SI, SAVED

FIND_LOOP_2:
    MOV DL, BYTE PTR[SI]
    MOV BYTE PTR [PATH_ + DI], DL
    INC DI
    INC SI
    CMP DL, 0
    JNE FIND_LOOP_2

    POP ES
    POP SI
    POP DI
    POP DX
    POP CX
    POP BX
    POP AX

```



```

    RET

PATH ENDP

ALLOCATE_MEMORY PROC NEAR
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH DI

    MOV DX, OFFSET DTA
    MOV AH, 1AH
    INT 21H

    MOV DX, OFFSET PATH_
    MOV AH, 4EH
    INT 21H

    JNC SUCCESSFUL_ALLOC

    CMP AX, 2
    JE CANT_FIND
    CMP AX, 3
    JE PATH_ERR

CANT_FIND:
    MOV DX, OFFSET CANT_FIND_ERROR2
    CALL WRITE_STRING
    JMP ALLOCATE_END

PATH_ERR:
    MOV DX, OFFSET PATH_ERROR2
    CALL WRITE_STRING
    JMP ALLOCATE_END

SUCCESSFUL_ALLOC:
    MOV DI, OFFSET DTA
    MOV DX, [DI + 1CH]
    MOV AX, [DI + 1AH]
    MOV BX, 10H
    DIV BX
    ADD AX, 1H
    MOV BX, AX
    MOV AH, 48H
    INT 21H
    MOV BX, OFFSET OVERLAY_ADDRESS
    MOV CX, 0000H
    MOV [BX], AX
    MOV [BX + 2], CX
    MOV DX, OFFSET NORMAL_ALLOC_END
    CALL WRITE_STRING

ALLOCATE_END:
    POP DI
    POP DX
    POP CX
    POP BX
    POP AX
    RET

```

```

ALLOCATE_MEMORY ENDP

LOAD PROC NEAR
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH DS
    PUSH ES

    MOV AX, DATA
    MOV ES, AX
    MOV BX, OFFSET OVERLAY_ADDRESS
    MOV DX, OFFSET PATH_
    MOV AX, 4B03H
    INT 21H

    JNC LOAD_SUCCESS
    CMP AX, 1
    JE E_1
    CMP AX, 2
    JE E_2
    CMP AX, 3
    JE E_3
    CMP AX, 4
    JE E_4
    CMP AX, 6
    JE E_6
    CMP AX, 8
    JE E_8
    CMP AX, 10
    JE E_10
E_1:
    MOV DX, OFFSET WRONG_NUMBER_ERROR
    CALL WRITE_STRING
    JMP LOAD_END
E_2:
    MOV DX, OFFSET CANT_FIND_ERROR
    CALL WRITE_STRING
    JMP LOAD_END
E_3:
    MOV DX, OFFSET PATH_ERROR
    CALL WRITE_STRING
    JMP LOAD_END
E_4:
    MOV DX, OFFSET OPEN_ERROR
    CALL WRITE_STRING
    JMP LOAD_END
E_6:
    MOV DX, OFFSET ACCESS_ERROR
    CALL WRITE_STRING
    JMP LOAD_END
E_8:
    MOV DX, OFFSET NOT_ENOUGH_MEM_ERROR
    CALL WRITE_STRING
    JMP LOAD_END

```

```

E_10:
    MOV DX, OFFSET ENVIRONMENT_ERROR
    CALL WRITE_STRING
    JMP LOAD_END

LOAD_SUCCESS:
    MOV DX, OFFSET NORMAL_END
    CALL WRITE_STRING

    MOV AX, WORD PTR OVERLAY_ADDRESS
    MOV ES, AX
    MOV WORD PTR OVERLAY_ADDRESS, 0
    MOV WORD PTR OVERLAY_ADDRESS + 2, AX
    CALL OVERLAY_ADDRESS
    MOV ES, AX
    MOV AH, 49H
    INT 21H

LOAD_END:
    POP SI
    POP DI
    POP DX
    POP CX
    POP BX
    POP AX
    RET
LOAD ENDP

FOR_OVERLAY PROC NEAR
    PUSH DX
    CALL PATH
    MOV DX, OFFSET PATH_
    CALL ALLOCATE_MEMORY
    CALL LOAD
    POP DX
    RET
FOR_OVERLAY ENDP

MAIN PROC FAR
    PUSH DX
    PUSH AX
    MOV AX, DATA
    MOV DS, AX

    MOV SAVED_PSP, ES

    CALL FREE_MEM
    CMP ERR_FLAG, 1
    JE MAIN_END

    MOV DX, OFFSET FILE_OVERLAY1
    CALL FOR_OVERLAY
    MOV DX, OFFSET FILE_OVERLAY2
    CALL FOR_OVERLAY

MAIN_END:
    MOV AH, 4CH

```

INT 21H

PROC_END:
MAIN ENDP
CODE ENDS
END MAIN