

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. 9381

Гурин С.Н.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры. Исследуется структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Для запуска вызываемого оверлейного модуля используется функция 4B03h прерывания int 21h. Все загрузочные и оверлейные модули находятся в одном каталоге.

В этой работе также рассматривается приложение, состоящее из нескольких модулей, поэтому все модули помещаются в один каталог и вызываются с использованием полного пути.

Порядок выполнения работы.

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.

Шаг 6. Занесите полученные результаты в виде скриншотов в отчет. Оформите отчет в соответствии с требованиями.

Выполнение работы.

Запустим программу из директории с оверлеями.

```
C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>LAB7.EXE
memory has been freed
allocation_mem was successful
load was successful

ovl_1 address:    0209

allocation_mem was successful
load was successful

ovl_2 address:    0209
```

Рис. 1

Теперь запустим так, что первый оверлей находится в другой директории.

```
C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>LAB7.EXE
memory has been freed
err: file not found(allocation_mem err)
err: file not found(load err)

allocation_mem was successful
load was successful

ovl_2 address:    0209
```

Рис. 2

Далее запустим так, что второй оверлей находится в другой директории.

```
C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>LAB7.EXE
memory has been freed
allocation_mem was successful
load was successful

ovl_1 address:    0209

err: file not found(allocation_mem err)
err: file not found(load err)
```

Рис. 3

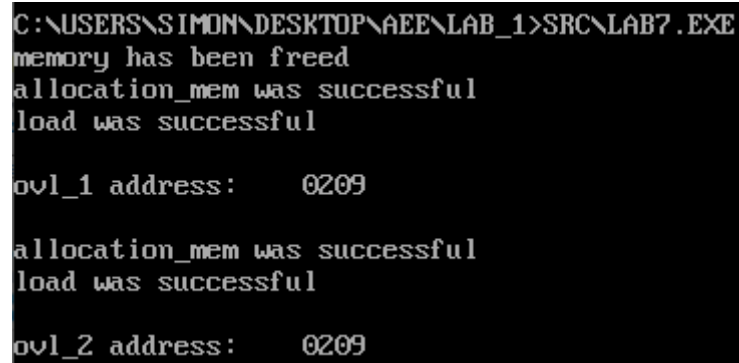
Так же запустим программу, только при условии, что оба оверлея находятся в разных директориях

```
C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>LAB7.EXE
memory has been freed
err: file not found(allocation_mem err)
err: file not found(load err)

err: file not found(allocation_mem err)
err: file not found(load err)
```

Рис. 4

Теперь запустим программу находясь в другой директории и при условии, что все оверлеи на месте.



```
C:\USERS\SIMON\DESKTOP\AEE\LAB_1>SRC\LAB7.EXE
memory has been freed
allocation_mem was successful
load was successful

ovl_1 address:    0209

allocation_mem was successful
load was successful

ovl_2 address:    0209
```

Рис. 5

Ответы на контрольные вопросы

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

После записи значений регистров в стек необходимо поместить регистр CS в регистр DS, так как адрес сегмента данных совпадает с адресом сегмента кода. Так же необходимо добавить 100h, т.к. изначально сегменты настроены на PSP.

Вывод

В ходе выполнения лабораторной работы был исследован принцип работы оверлейных структур и принцип их загрузки.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab7.asm

DATA	SEGMENT	
FILE1	DB	"OVERLAY1.OVL", 0
FILE2	DB	"OVERLAY2.OVL", 0
PROGRAM	DW	0
DTA_MEM	DB	43 DUP(0)
MEM_FLAG	DB	0
CL_POS	DB	128 DUP(0)
OVLS_ADDR	DD	0
KEEP_PSP	DW	0
EOF	DB	0DH, 0AH, '\$'
STR_MCB_CRASH_ERROR	DB	'ERR: MCB CRASHED', 0DH, 0AH, '\$'
STR_NO_MEMORY_ERROR	DB	'ERR: THERE IS NOT ENOUGH MEMORY TO EXECUTE THIS FUNCTION', 0DH, 0AH, '\$'
STR_ADDR_ERROR	DB	'ERR: INVALID MEMORY ADDRESS', 0DH, 0AH, '\$'
STR_FREE_MEMORY	DB	'MEMORY HAS BEEN FREED' , 0DH, 0AH, '\$'
STR_FN_ERROR	DB	'ERR: FUNCTION DOESNT EXIST', 0DH, 0AH, '\$'
STR_FILE_ERROR	DB	'ERR: FILE NOT FOUND(LOAD ERR)', 0DH, 0AH, '\$'
STR_ROUTE_ERROR	DB	'ERR: ROUTE NOT FOUND(LOAD ERR)', 0DH, 0AH, '\$'
STR_FILES_ERROR	DB	'ERR: YOU OPENED TOO MANY FILES', 0DH, 0AH, '\$'
STR_ACCESS_ERROR	DB	'ERR: NO ACCESS', 0DH, 0AH, '\$'
STR_MEM_ERROR	DB	'ERR: INSUFFICIENT MEMORY', 0DH, 0AH, '\$'
STR_ENVS_ERROR	DB	'ERR: WRONG STRING OF ENVIRONMENT ', 0DH, 0AH, '\$'
STR_NORMAL_END	DB	'LOAD WAS SUCCESSFUL', 0DH, 0AH, '\$'
STR_ALLOC_MEM_END	DB	'ALLOCATION_MEM WAS SUCCESSFUL', 0DH, 0AH, '\$'

```

STR_ALL_FILE_ERROR    DB    'ERR: FILE NOT FOUND (ALLOCATION_MEM ERR)' ,
0DH, 0AH, '$'
STR_ALL_ROUTE_ERROR DB    'ERR: ROUTE NOT FOUND (ALLOCATION_MEM ERR)' ,
0DH, 0AH, '$'
END_DATA              DB          0
DATA                  ENDS

```

```

SSTACK                SEGMENT    STACK
                        DW          128 DUP(?)
SSTACK                ENDS

```

```

CODE                  SEGMENT

```

```

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

```

```

PRINT_STR             PROC
                        PUSH AX
                        MOV  AH, 09H
                        INT  21H
                        POP  AX
                        RET

```

```

PRINT_STR             ENDP

```

```

FREE_MEMORY           PROC
                        PUSH AX
                        PUSH BX
                        PUSH CX
                        PUSH DX

                        MOV  AX, OFFSET END_DATA
                        MOV  BX, OFFSET __END
                        ADD  BX, AX

                        MOV  CL, 4
                        SHR  BX, CL
                        ADD  BX, 2BH
                        MOV  AH, 4AH
                        INT  21H

```

```

        JNC  FREE_MEM_END
        MOV  MEM_FLAG, 1

_MCB_CRASH:
        CMP  AX, 7
        JNE  NO_MEMORY
        MOV  DX, OFFSET STR_MCB_CRASH_ERROR

        CALL PRINT_STR

        JMP  END_FREE
NO_MEMORY:
        CMP  AX, 8
        JNE  _ADDR
        MOV  DX, OFFSET STR_NO_MEMORY_ERROR

        CALL PRINT_STR

        JMP  END_FREE
_ADDR:
        CMP  AX, 9
        MOV  DX, OFFSET STR_ADDR_ERROR

        CALL PRINT_STR

        JMP  END_FREE
FREE_MEM_END:
        MOV  MEM_FLAG, 1
        MOV  DX, OFFSET STR_FREE_MEMORY

        CALL PRINT_STR
END_FREE:
        POP  DX
        POP  CX
        POP  BX
        POP  AX
        RET

```

FREE_MEMORY

ENDP

LOAD_PROC

PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

PUSH DS

PUSH ES

MOV AX, DATA

MOV ES, AX

MOV BX, OFFSET OVLS_ADDR

MOV DX, OFFSET CL_POS

MOV AX, 4B03H

INT 21H

JNC _LOADS

_FN_ERR:

CMP AX, 1

JNE FILE_ERR

MOV DX, OFFSET EOF

CALL PRINT_STR

MOV DX, OFFSET STR_FN_ERROR

PUSH AX

MOV AH, 09H

INT 21H

POP AX

JMP _LOADE

FILE_ERR:

CMP AX, 2

JNE ROUTE_ERR


```

MOV     DX, OFFSET STR_FILE_ERROR

PUSH    AX
MOV     AH, 09H
INT     21H
POP     AX

JMP     _LOADE
ROUTE_ERR:
CMP     AX, 3
JNE     _FILES_ERR
MOV     DX, OFFSET EOF

CALL    PRINT_STR

MOV     DX, OFFSET STR_ROUTE_ERROR

CALL    PRINT_STR

JMP     _LOADE
_FILES_ERR:
CMP     AX, 4
JNE     _ACCESS_ERR
MOV     DX, OFFSET STR_FILES_ERROR

PUSH    AX
MOV     AH, 09H
INT     21H
POP     AX

JMP     _LOADE
_ACCESS_ERR:
CMP     AX, 5
JNE     _MEM_ERR
MOV     DX, OFFSET STR_ACCESS_ERROR

CALL    PRINT_STR

```

```

                                JMP    _LOADE
MEM_ERR:
                                CMP    AX, 8
                                JNE     ENVS_ERROR
                                MOV     DX, OFFSET STR_MEM_ERROR

                                CALL    PRINT_STR

                                JMP     _LOADE
ENVS_ERROR:
                                CMP     AX, 10
                                MOV     DX, OFFSET STR_ENVS_ERROR

                                CALL    PRINT_STR

                                JMP     _LOADE

_LOADS:
                                MOV     DX, OFFSET STR_NORMAL_END

                                CALL    PRINT_STR

                                MOV     AX, WORD PTR OVLS_ADDR
                                MOV     ES, AX
                                MOV     WORD PTR OVLS_ADDR, 0
                                MOV     WORD PTR OVLS_ADDR+2, AX

                                CALL    OVLS_ADDR
                                MOV     ES, AX
                                MOV     AH, 49H
                                INT     21H

_LOADE:
                                POP     ES
                                POP     DS
                                POP     DX
                                POP     CX
                                POP     BX

```

```

                                POP    AX
                                RET
LOAD_PROC                      ENDP

PATH                            PROC

                                PUSH   AX
                                PUSH   BX
                                PUSH   CX
                                PUSH   DX
                                PUSH   DI
                                PUSH   SI
                                PUSH   ES

                                MOV     PROGRAM, DX

                                MOV     AX, KEEP_PSP
                                MOV     ES, AX
                                MOV     ES, ES:[2CH]
                                MOV     BX, 0

                                FINDZ:

                                INC     BX
                                CMP     BYTE PTR ES:[BX-1], 0
                                JNE     FINDZ

                                CMP     BYTE PTR ES:[BX+1], 0
                                JNE     FINDZ

                                ADD     BX, 2
                                MOV     DI, 0

                                _LOOP:

                                MOV     DL, ES:[BX]
                                MOV     BYTE PTR [CL_POS+DI], DL
                                INC     DI
                                INC     BX
                                CMP     DL, 0
                                JE       _END_LOOP

```

```

                                CMP    DL, '\'
                                JNE     _LOOP
                                MOV     CX, DI
                                JMP     _LOOP
_END_LOOP:
                                MOV     DI, CX
                                MOV     SI, PROGRAM

_FN:
                                MOV     DL, BYTE PTR [SI]
                                MOV     BYTE PTR [CL_POS+DI], DL
                                INC     DI
                                INC     SI
                                CMP     DL, 0
                                JNE     _FN

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

PATH                            ENDP

ALLOCATION_MEM                  PROC
                                PUSH    AX
                                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    _ALL_SUCCESS

_ALLFILE_ERR:
    CMP    AX, 2
    JE     _ALLROUTE_ERR
    MOV    DX, OFFSET STR_ALL_FILE_ERROR

    CALL PRINT_STR

    JMP    _ALL_END

_ALLROUTE_ERR:
    CMP    AX, 3
    MOV    DX, OFFSET STR_ALL_ROUTE_ERROR

    CALL PRINT_STR

    JMP    _ALL_END

_ALL_SUCCESS:
    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 OVLS_ADDR, AX

```

```

MOV     DX, OFFSET STR_ALLOC_MEM_END

CALL PRINT_STR

    _ALL_END:
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        RET

ALLOCATION_MEM      ENDP

BEGIN_OVL          PROC
    PUSH DX
    CALL PATH
    MOV  DX, OFFSET CL_POS
    CALL ALLOCATION_MEM
    CALL LOAD_PROC
    POP  DX
    RET

BEGIN_OVL          ENDP

MAIN               PROC FAR
    PUSH DS
    XOR  AX, AX
    PUSH AX
    MOV  AX, DATA
    MOV  DS, AX
    MOV  KEEP_PSP, ES
    CALL FREE_MEMORY
    CMP  MEM_FLAG, 0
    JE   _END

    MOV  DX, OFFSET FILE1
    CALL BEGIN_OVL
    MOV  DX, OFFSET EOF

    CALL PRINT_STR

```

```

                                MOV  DX, OFFSET FILE2
                                CALL  BEGIN_OVL

                                _END:

                                XOR   AL, AL
                                MOV   AH, 4CH
                                INT   21H

MAIN                            ENDP

                                __END:

CODE                            ENDS

                                END   MAIN

```