**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. 7381 | Минуллин М.А. |
| Преподаватель | Ефремов М.А. |

Санкт-Петербург

2019

**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 21h. Все загрузочные и оверлейные модули находятся в одном каталоге.

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

**Необходимые сведения для составления программы.**

Для организации программы, имеющей оверлейную структуру, используется функция 4B03h прерывания 21h. Эта функция позволяет в отведённую область памяти, начинающуюся с адреса сегмента, загрузить программу, находящуюся в файле на диске. Передача управления загруженной программе этой функцией не осуществляется и префикс сегмента программы (PSP) не создаётся.

Если флаг переноса после выполнения функции, то произошли ошибки и регистр содержит код ошибки. Значение регистра характеризует следующие ситуации, представленные в табл. 1.

Таблица 1 – Возможные ошибки при выполнении функции 4B03h.

|  |  |
| --- | --- |
| Код ошибки | Описание |
| 1 | Несуществующая функция |
| 2 | Файл не найден |
| 3 | Маршрут не найден |
| 4 | Слишком много открытых файлов |
| 5 | Нет доступа |
| 8 | Мало памяти |
| 10 | Неправильная среда |

Если флаг переноса , то оверлей загружен в память.

Перед загрузкой оверлея вызывающая программа должна освободить память по функции 4Ah прерывания 21h. Затем определить размер оверлея. Это можно сделать с помощью функции 4Eh прерывания 21h. Перед обращением к функции необходимо определить область памяти размером в 43 байта под буфер DTA, которую функция заполнит, если файл будет найден.

Функция использует следующие параметры: – значение байта атрибутов, которое для файла имеет значение 0; – указатель на путь к файлу, который записывается в формате строки ASCIIZ.

Если флаг переноса после выполнения функции, то произошли ошибки и регистр содержит код ошибки. Значение регистра характеризует ситуации, представленные в табл. 2.

Таблица 2 – Возможные ошибки при выполнении функции 4Eh.

|  |  |
| --- | --- |
| Код ошибки | Описание |
| 2 | Файл не найден |
| 3 | Маршрут не найден |

Если , то в области памяти буфера DTA со смещением 1Ah будет находиться младшее слово размера файла, а в слове со смещением 1Ch – старшее слово размера памяти в байтах.

Полученный размер файла следует перевести в параграфы, причём следует взять большое целое числа параграфов. Затем необходимо отвести память с помощью функции 48h прерывания 21h. После этого необходимо сформировать параметры для функции 4B03h и выполнить её.

После отработки оверлея необходимо освободить память с помощью функции 49h прерывания 21h.

Оверлейный сегмент не является загрузочным модулем типов .COM или .EXE. Он представляет собой кодовый сегмент, который оформляется в ассемблере как функция с точкой входа по адресу 0 и возврат осуществляется командой retf. Это необходимо сделать, потому что возврат управления должен быть осуществлён в программу, выполняющую оверлейный сегмент. Если использовать функции выхода 4Ch прерывания 21h, то программа закончит свою работу.

**Ход работы.**

Был написан и отлажен программный модуль типа .EXE, выполняющий функции:

Освобождение памяти для загрузки оверлеев.

Чтение размера файла оверлея и запрашивание память в объёме, достаточном для его загрузки.

Загрузка файла оверлейного сегмента.

Освобождение памяти, отведённой для оверлейного сегмента.

Повтор действий 1-4 для следующего оверлейного сегмента.

Были написаны и отлажены оверлейные сегменты, выводящие адрес сегмента, в которые они загружены (см. рис. 1).

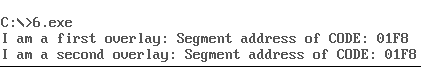


Рисунок 1 – результат работы программы.

Приложение было запущено из другого каталога. Результат работы приложения виден на рис. 2.

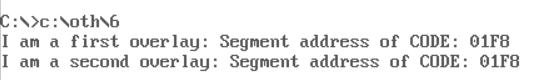


Рисунок 2 – запуск приложения из другого каталога.

Приложение было запущено в случае, когда одного оверлея нет в каталоге (см. рис. 3).

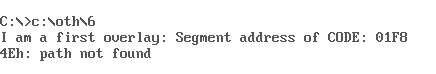


Рисунок 3 – одного из оверлеев нет в каталоге.

**Контрольные вопросы.**

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

О: при построении .COM модуля линковщик из-за директивы org 100h смещает все метки на 256 байт вперёд. Смещение 100h требуется, так как MS DOS загружает сегмент .COM модуля не от начала сегмента в памяти, а по смещению 100h (первые 100h отводятся под PSP).

При загрузке .COM модуля в качестве оверлея юCOM-сегмент загружается без смещения 100h. Следовательно, возникает проблема в том, что все метки и смещения, посчитанные линковщиком, «съезжают» на 100h вперёд.

Для решения этой проблемы нужно вместо вызова функции в .COM файле по нулевому смещению вызвать функцию по смещению 100h и вместо реального сегментного адреса указывать сегментный адрес на 10h меньший исходного. Тогда .COM модуль будет работать с правильными смещениями.

**Выводы.**

В ходе выполнения лабораторной работы было проведено исследование возможности построения загрузочного модуля оверлейной структуры. Была исследована структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Для запуска вызываемого оверлейного модуля использовалась функция 4B03h прерывания 21h. Все загрузочные и оверлейные модули находились в одном каталоге.

Было разработано приложение, состоящее из нескольких модулей, поэтому все модули помещались в один каталог и вызывались с помощью относительного пути.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ ТЕКСТ .EXE МОДУЛЯ**

.286

SSEG segment stack

db 100h dup(?)

SSEG ends

DATA segment

    CB7 DB '04h: control block crached',0AH,0DH,'$'

CB8 DB '04h: memory not enough ',0AH,0DH,'$'

CB9 DB '04h: wrong address of control block',0AH,0DH,'$'

NL1 DB '4B03: wrong function number', 0AH,0DH, '$'

NL2 DB '4B03: file not found', 0AH,0DH, '$'

NL3 DB '4B03: path error', 0AH,0DH, '$'

NL4 DB '4B03: too many opened files', 0AH,0DH, '$'

NL5 DB '4B03: not access', 0AH,0DH, '$'

NL8 DB '4B03: memory not enough', 0AH,0DH, '$'

NL10 DB '4B03: wrong environment', 0AH,0DH, '$'

SZ2 DB '4Eh: file not found', 0AH,0DH, '$'

SZ3 DB '4Eh: path not found', 0AH,0DH, '$'

AM DB '48h: memory not allocate', 0ah, 0dh, '$'

DAM DB '49h: memory not deallocate', 0ah, 0dh, '$'

;------------ PARAM\_BLOCK ------------------;

PARAMS dw 0 , 0 ; сегментный адрес загрузки оверлея

;--------- END OF PARAM\_BLOCK --------------;

PATH\_PROMT db 81h dup(0)

DTA\_BUFFER db 43 dup(?)

DATA ends

CODE segment

    assume CS:CODE, DS:DATA, ES:DATA, SS:SSEG

FREE\_MEMORY proc near

pusha

push ds

push es

mov dx, cs:KEEP\_PSP

mov es, dx

mov bx, offset last\_byte

shr bx, 4

inc bx

add bx, CODE

sub bx, cs:KEEP\_PSP

mov ah, 4Ah

int 21h

jnc free\_memory\_success

jmp free\_memory\_error

free\_memory\_success:

pop es

pop ds

popa

ret

free\_memory\_error:

next7:

cmp ax, 7

jne next8

mov dx, offset CB7

jmp print\_label

next8:

cmp ax, 8

jne next9

mov dx, offset CB8

jmp print\_label

next9:

mov dx, offset CB9

print\_label:

call print

mov ah, 4ch

mov al,0

int 21h

FREE\_MEMORY endp

PRINT\_NOT\_LOAD\_ERROR proc near

nextNL1:

cmp ax, 1

jne nextNL2

mov dx, offset NL1

jmp print\_label\_NL

nextNL2:

cmp ax, 2

jne nextNL3

mov dx, offset NL2

jmp print\_label\_NL

nextNL3:

cmp ax, 3

jne nextNL4

mov dx, offset NL3

jmp print\_label\_NL

nextNL4:

cmp ax, 4

jne nextNL5

mov dx, offset NL4

jmp print\_label\_NL

nextNL5:

cmp ax, 5

jne nextNL8

mov dx, offset NL5

jmp print\_label\_NL

nextNL8:

cmp ax, 8

jne nextNL10

mov dx, offset NL8

jmp print\_label\_NL

nextNL10:

mov dx, offset NL10

print\_label\_NL:

call print

xor AL,AL

    mov AH,4Ch

    int 21H

PRINT\_NOT\_LOAD\_ERROR endp

PRINT proc near ; dx = OFFSET TO STR

pusha

push ds

mov ax, DATA

mov ds, ax

mov ah, 09h

int 21h

pop ds

popa

ret

PRINT endp

init\_child\_path1 proc near

pusha

push es

push ds

mov dx, cs:KEEP\_PSP

mov ds, dx

mov dx, DATA

mov es, dx

mov dx, ds:[2ch]

mov ds, dx

mov si, 0

cicl:

cmp word ptr ds:[si], 0

je break

inc si

jmp cicl

break:

add si, 4

mov di, offset PATH\_PROMT

cicl2:

cmp byte ptr ds:[si], 0

je break2

mov al, ds:[si]

mov byte ptr es:[di], al

inc si

inc di

jmp cicl2

break2:

mov byte ptr es:[di-1], 'L'

mov byte ptr es:[di-2], 'V'

mov byte ptr es:[di-3], 'O'

mov byte ptr es:[di-4], '.'

mov byte ptr es:[di-5], '1'

pop ds

pop es

popa

ret

init\_child\_path1 endp

init\_child\_path2 proc near

pusha

push es

push ds

mov dx, cs:KEEP\_PSP

mov ds, dx

mov dx, DATA

mov es, dx

mov dx, ds:[2ch]

mov ds, dx

mov si, 0

cicl\_2:

cmp word ptr ds:[si], 0

je break\_2

inc si

jmp cicl\_2

break\_2:

add si, 4

mov di, offset PATH\_PROMT

cicl\_22:

cmp byte ptr ds:[si], 0

je break\_22

mov al, ds:[si]

mov byte ptr es:[di], al

inc si

inc di

jmp cicl\_22

break\_22:

mov byte ptr es:[di-1], 'L'

mov byte ptr es:[di-2], 'V'

mov byte ptr es:[di-3], 'O'

mov byte ptr es:[di-4], '.'

mov byte ptr es:[di-5], '2'

pop ds

pop es

popa

ret

init\_child\_path2 endp

GET\_SIZE\_OF\_FILE proc near

push dx

push cx

push ds

mov dx, DATA

mov ds, dx

mov dx, offset PATH\_PROMT

mov cx, 0

mov ah, 4Eh

int 21h

jnc sizeOfFile\_success

nextSZ2:

cmp ax, 2

jne nextSZ3

mov dx, offset SZ2

jmp print\_label\_SZ

nextSZ3:

mov dx, offset SZ3

print\_label\_SZ:

call PRINT

mov ax, 0

pop ds

pop cx

pop dx

ret

sizeOfFile\_success:

mov ax, [offset DTA\_BUFFER+1Ah]

mov dx, 0

shr ax, 4

inc ax

pop ds

pop cx

pop dx

ret

GET\_SIZE\_OF\_FILE endp

ALLOC\_MEM proc near

push bx

mov bx, ax

mov ah, 48h

int 21h

jnc continue\_ALLOC

mov dx, offset AM

call print

mov ah, 4ch

int 21h

continue\_ALLOC:

pop bx

ret

ALLOC\_MEM endp

DEALLOC\_MEM proc near

pusha

push dx

push es

mov dx, DATA

mov ds, dx

mov ah, 49h

mov dx, ds:PARAMS+2

mov es, dx

int 21h

jnc del\_next

mov dx, offset DAM

call PRINT

mov ah, 4ch

mov al, 0

int 21h

del\_next:

pop es

pop dx

popa

ret

DEALLOC\_MEM endp

LOAD\_OVL proc near

pusha

push ds

push es

mov bx, DATA

mov ds, bx

mov es, bx

mov ds:PARAMS + 2, ax

mov bx, offset PARAMS + 2

mov dx, offset PATH\_PROMT

mov ax, 4B03h

int 21h

jnc continue

call PRINT\_NOT\_LOAD\_ERROR

continue:

pop es

pop ds

popa

ret

LOAD\_OVL endp

RUN\_OVL proc near

pusha

push ds

push es

mov dx, DATA

mov ds, dx

push cs

mov ax, offset exit\_of\_overlay

push ax

jmp dword ptr ds:PARAMS

exit\_of\_overlay:

pop es

pop ds

popa

ret

RUN\_OVL endp

START:

    push ds

    sub ax, ax

    push ax

mov cs:KEEP\_PSP, ds

call FREE\_MEMORY

call init\_child\_path1

call GET\_SIZE\_OF\_FILE

cmp ax, 0

je next\_ovl

call ALLOC\_MEM

call LOAD\_OVL

call RUN\_OVL

call DEALLOC\_MEM

next\_ovl:

call init\_child\_path2

call GET\_SIZE\_OF\_FILE

cmp ax, 0

je exit\_of\_prog

call ALLOC\_MEM

call LOAD\_OVL

call RUN\_OVL

call DEALLOC\_MEM

exit\_of\_prog:

mov ah, 4ch

mov al, 0

int 21h

KEEP\_PSP dw 0h

last\_byte:

CODE ends

END START

**ПРИЛОЖЕНИЕ Б**

**ИСХОДНЫЙ ТЕКСТ ПЕРВОГО ОВЕРЛЕЙНОГО МОДУЛЯ**

CODE segment

    assume  CS:CODE, DS:CODE

START:

    push    ds

push    cs

pop     ds

    mov     di, offset ADDR\_STRING + 50

mov     ax, cs

    call    WRD\_TO\_HEX

    mov     dx, offset ADDR\_STRING

    mov     ah, 09h

int     21h

    pop     ds

    retf

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

    push    cx

    mov     ah, al

    call    TETR\_TO\_HEX

    xchg    al, ah

    mov     cl, 4

    shr     al, cl

    call    TETR\_TO\_HEX

    pop     cx

    ret

BYTE\_TO\_HEX endp

WRD\_TO\_HEX proc near

    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

    ADDR\_STRING     DB 'I am a first overlay: Segment address of CODE: ',0DH,0AH,'$'

CODE ends

end START

**ПРИЛОЖЕНИЕ В**

**ИСХОДНЫЙ ТЕКСТ ВТОРОГО ОВЕРЛЕЙНОГО МОДУЛЯ**

CODE segment

    assume CS:CODE, DS:CODE

START:

    push ds

push cs

pop ds

    mov di, offset ADDR\_STRING + 51

mov ax, cs

    call WRD\_TO\_HEX

    mov dx, offset ADDR\_STRING

    mov ah, 09h

    int 21h

    pop ds

    retf

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

push cx

mov ah, al

call TETR\_TO\_HEX

xchg al, ah

mov cl, 4

shr al, cl

call TETR\_TO\_HEX

pop cx

ret

BYTE\_TO\_HEX endp

WRD\_TO\_HEX proc near

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

    ADDR\_STRING     DB 'I am a second overlay: Segment address of CODE: ',0DH,0AH,'$'

CODE ends

end START