**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 |  | Орлов Д.С. |
| 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.

**Постановка задачи.**

**Шаг 1**. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .ЕХЕ, 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.**

**1.**  Был написан и отлажен программный модуль типа .ЕХЕ, 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.

**2.** Были написаны и отлажены оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

**3.**  Было запущено отлаженное приложение. Оверлейные сегменты загружаются с одного и того же адреса, перекрывая друг друга.



Рисунок 1 - Вывод программы lb7.exe

**4.**  Приложение было успешно выполнено из другого каталога.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 - Запуск программы из другого каталога

**5.**  Было запущено приложение, когда оверлея №2 нет в каталоге. Приложение закончилось аварийно.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 - Результат запуска программы на 5 шаге

**Ответы на вопросы.**

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

Необходимо учитывать смещение 100h, так как в .COM модуле присутствует PSP. После записи значений регистров в стек, нужно положить значение регистра CS в DS, так как адрес сегмента данных совпадает с сегментом кода.

**Вывод.**

В результате выполнения работы были исследованы возможности построения загрузочного модуля оверлейной структуры.

**ПРИЛОЖЕНИЕ А**

**КОД ПРОГРАММЫ**

lb7.asm

code segment

assume CS:code, DS:data

println proc

push AX

push DX

mov AH, 09h

int 21h

mov AH, 02h

mov DL, 0Ah

int 21h

mov DL, 0Dh

int 21h

pop DX

pop AX

ret

println endp

num2dec proc

push AX

push BX

push CX

push DX

push SI

mov BX, 10

xor CX, CX

xor DX, DX

div10: div BL

mov DL, AH

push DX

xor AH, AH

inc CX

xor DX, DX

cmp AL, 0

jne div10

loop10: pop DX

xor DH, DH

add DL, 30h

mov byte ptr [SI], DL

inc SI

loop loop10

pop SI

pop DX

pop CX

pop BX

pop AX

ret

num2dec endp

free proc

push AX

push BX

push DX

xor DX, DX

mov AX, offset endprog

add AX, offset enddata

add AX, 300h

mov BX, 16

div BX

mov BX, AX

inc BX

mov AH, 4Ah

int 21h

jnc endf

cmp AX, 7

je efree7

cmp AX, 8

je efree8

cmp AX, 9

je efree9

jmp efreeU

efree7: mov DX, offset ferr7

jmp eprint

efree8: mov DX, offset ferr8

jmp eprint

efree9: mov DX, offset ferr9

jmp eprint

efreeU: mov DX, offset unknown

eprint: call println

mov flag, 1

endf: pop DX

pop BX

pop AX

ret

free endp

setp proc

push AX

mov AX, ES:[2Ch]

mov param, AX

mov param + 2, ES

mov param + 4, 80h

pop AX

ret

setp endp

getpath proc

push DX

push DI

push SI

push ES

xor DI, DI

mov ES, ES:[2Ch]

mov DL, ES:[DI]

jmp check

nextc: inc DI

mov DL, ES:[DI]

check: cmp DL, 00h

jne nextc

inc DI

mov DL, ES:[DI]

cmp DL, 00h

jne nextc

xor SI, SI

add DI, 3

getc: mov DL, ES:[DI]

cmp DL, 00h

je getf

mov byte ptr opath1[SI], DL

mov byte ptr opath2[SI], DL

inc DI

inc SI

jmp getc

getf: dec SI

mov DL, opath1[SI]

cmp DL, '\'

jne getf

inc SI

xor DI, DI

addp: mov DL, oname1[DI]

mov DH, oname2[DI]

cmp DL, '$'

je endg

mov opath1[SI], DL

mov opath2[SI], DH

inc DI

inc SI

jmp addp

endg: mov opath1[SI], 00h

mov opath2[SI], 00h

pop ES

pop SI

pop DI

pop DX

ret

getpath endp

callovl proc

push AX

push BX

push CX

push DX

push SI

mov AX, 1A00h

mov DX, offset DTA

int 21h

mov AX, 4E00h

mov CX, 0

mov DX, file

int 21h

jnc ovl

cmp AX, 3

je esize3

cmp AX, 2

je esize2

esize2: mov DX, offset serr2

jmp errend

esize3: mov DX, offset serr3

jmp errend

ovl: mov SI, offset DTA

mov BX, [SI + 1Ah]

mov CL, 4

shr BX, CL

mov AX, [SI + 1Ch]

mov CL, 12

shl AX, CL

add BX, AX

add BX, 1

mov DX, file

mov AX, 4800h

int 21h

mov oblock, AX

push DS

push ES

mov keepSS, SS

mov keepSP, SP

mov AX, DS

mov ES, AX

mov BX, offset oblock

mov DX, file

mov AX, 4B03h

int 21h

mov DX, keepSP

mov SP, DX

mov SS, keepSS

pop ES

pop DS

jnc ok

cmp AX, 1

je ecall1

cmp AX, 2

je ecall2

cmp AX, 3

je ecall3

cmp AX, 4

je ecall4

cmp AX, 5

je ecall5

cmp AX, 8

je ecall8

cmp AX, 10

je ecallA

jmp ecallU

ecall1: mov DX, offset cerr1

jmp errend

ecall2: mov DX, offset cerr2

jmp errend

ecall3: mov DX, offset cerr3

jmp errend

ecall4: mov DX, offset cerr4

jmp errend

ecall5: mov DX, offset cerr5

jmp errend

ecall8: mov DX, offset cerr8

jmp errend

ecallA: mov DX, offset cerrA

jmp errend

ecallU: mov DX, offset unknown

jmp errend

ok: push ES

mov AX, oblock

mov ES, AX

mov word ptr ovlbeg + 2, AX

call DS:[ovlbeg]

mov ES, AX

mov AX, 4900h

int 21h

pop ES

jmp oend

errend: call println

mov flag, 1

oend: pop SI

pop DX

pop CX

pop BX

pop AX

ret

callovl endp

main proc far

mov AX, data

mov DS, AX

call free

cmp flag, 0

jne fin

call setp

call getpath

mov DX, offset opath1

mov file, DX

call callovl

cmp flag, 0

jne fin

mov DX, offset opath2

mov file, DX

call callovl

; --- End ---

fin: mov AX, 4C00h

int 21h

main endp

endprog:

code ends

data segment

param dw 7 dup (0)

DTA db 43 dup (0)

oname1 db 'OVERLAY1.OVL$'

oname2 db 'OVERLAY2.OVL$'

opath1 db 64 dup (0), '$'

opath2 db 64 dup (0), '$'

file dw 0

oblock dw 0

flag db 0

keepSS dw 0

keepSP dw 0

unknown db 'Unknown error$'

ferr7 db 'Error: memory control block destroyed$'

ferr8 db 'Error: not enough memory to execute the function$'

ferr9 db 'Error: invalid memory block address$'

serr2 db 'Error: file could not be find$'

serr3 db 'Error: path could not be find$'

cerr1 db 'Error: function is not exists$'

cerr2 db 'Error: file could not be found$'

cerr3 db 'Error: path could not be found$'

cerr4 db 'Error: too many open files$'

cerr5 db 'Error: no access$'

cerr8 db 'Error: no memory$'

cerrA db 'Error: wrong environment string$'

ovlbeg dd 0

enddata db ?

data ends

stack segment stack

db 256 dup (?)

stack ends

end main