

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

Мосин К.К.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры.

Задание.

Шаг 1. Написать и отладить модуль типа .EXE, который выполняет следующие 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) Действия 1)-4) выполняются для следующего оверлейного сегмента

Шаг 2. Написать и отладить оверлейные сегменты.

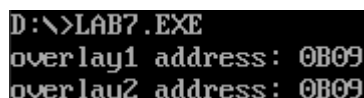
Шаг 3. Запуск отлаженной программы.

Шаг 4. Запуск отлаженной программы из другого каталога.

Шаг 5. Запуск приложения без оверлея в каталоге.

Выполнение работы.

Был разработан модуль типа .EXE. Пример работы проиллюстрирован на изображении 1. Также был произведен запуск отлаженной программы из другого каталога. Результат теста представлен на рисунке 2. Была произведена попытка запуска программы, когда один из оверлейных модулей находился в другом каталоге. Пример показан на изображении 3.



```
D:\>LAB7.EXE
overlay1 address: 0B09
overlay2 address: 0B09
```

Рис. 1 - Пример работы .EXE модуля

```
Z:\>D:\LAB7.EXE  
overlay1 address: 0172  
overlay2 address: 0172
```

Рис. 2 - Пример работы .EXE модуля при запуске из другого каталога

```
D:\>LAB7.EXE  
file not found  
overlay2 address: 0B09
```

Рис. 3 - Пример работы .EXE модуля с одной оверлейной программой

Контрольные вопросы.

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

Необходимо учитывать смещение 100h и вычитать его из адреса данных.

Выводы.

В ходе выполнения лабораторной работы были построены загрузочные модули оверлейной структуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab7.asm

```
stack_ segment stack
```

```
dw 256 dup(?)
```

```
stack_ ends
```

```
data_ segment
```

```
overlay_size_data db 43 dup(0),'$'
```

```
file_path dw 50 dup(0)
```

```
file_name dw 0
```

```
overlay_name1 db 'OVL1.OVL',0
```

```
overlay_name2 db 'OVL2.OVL',0
```

```
overlay_ptr dd 0
```

```
keep_psp dw 0
```

```
exit_code db 'Program finished with code: $'
```

```
error_1 db 'non-existent function',0dh,0ah,'$'
```

```
error_2 db 'file not found',0dh,0ah,'$'
```

```
error_3 db 'path not found',0dh,0ah,'$'
```

```
error_4 db 'many files are open',0dh,0ah,'$'
```

```
error_5 db 'no access',0dh,0ah,'$'
```

```
error_8 db 'not enough memory',0dh,0ah,'$'
```

```
error_10 db 'wrong environment',0dh,0ah,'$'
```

```
error_free_memory db 'fail free memory',0dh,0ah,'$'
```

```
exec_parameter_block dw 0
```

```
db 0
```

```
db 0
```

```
db 0
```

data_ ends

code_ segment

assume cs:code_, ds:data_, ss:stack_

print proc near

push ax

mov ah,09h

int 21h

pop ax

ret

print endp

word_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

xor ah,ah

call byte_to_hex

mov [di],ah

dec di

mov [di],al

pop bx

ret

word_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
```

```
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
```

```
free_memory proc far
mov bx,offset main_end_prt
mov ax,es
sub bx,ax
add bx,950h
mov ah,4ah
int 21h
```

```
jc free_memory_error
ret
free_memory_error:
push dx
mov dx,offset error_free_memory
call print
pop dx
ret
free_memory endp
```

```
find_path proc
```

```
push ax
push bx
push cx
push dx
push di
push si
push es
```

```
mov file_name,dx
```

```
mov es,es:[2ch]
```

```
mov bx,0
```

```
loop_:
```

```
cmp byte ptr es:[bx],00h
```

```
jne next_byte
```

```
cmp byte ptr es:[bx+1],00h
```

```
jne next_byte
```

```
jmp path_found
```

next_byte:

inc bx

jmp loop_

path_found:

add bx,4

xor si, si

lea di,file_path

read:

mov dl,es:[bx+si]

cmp byte ptr es:[bx+si],0

je insert_file_name

mov [di],dl

inc di

inc si

jmp read

insert_file_name:

sub di,8

mov cx,8

mov si,file_name

copy:

mov al,[si]

mov [di],al

inc si

inc di

loop copy

mov [di],byte ptr '\$'


```
pop es
pop si
pop di
pop dx
pop cx
pop bx
pop ax
ret
find_path endp
```

```
overlay_size proc near
```

```
push ax
push bx
push cx
push dx
```

```
push dx
mov dx,offset overlay_size_data
mov ah,1ah
int 21h
pop dx
```

```
mov cx,0
mov ah,4eh
int 21h
```

```
push di
mov di,offset overlay_size_data
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 overlay_ptr,ax

pop dx

pop cx

pop bx

pop ax

ret

overlay_size endp

overlay_execute proc near

push ax

push bx

push cx

push dx

push ds

push es

mov ax,data_

mov es,ax

mov bx,offset overlay_ptr

mov dx,offset file_path

mov ax,4b03h

int 21h

jnc execution

jmp overlay_execute_error

execution:

mov ax,word ptr overlay_ptr

mov es,ax

mov word ptr overlay_ptr,0

mov word ptr overlay_ptr+2,ax

call overlay_ptr

mov es,ax

mov ah,49h

int 21h

jmp execution_is_over

overlay_execute_error:

cmp ax,1

je print_overlay_error1

cmp ax,2

je print_overlay_error2

cmp ax,3

je print_overlay_error3

```
cmp ax,4
je print_overlay_error4
cmp ax,5
je print_overlay_error5
cmp ax,8
je print_overlay_error8
cmp ax,10
je print_overlay_error10
```

```
print_overlay_error1:
mov dx,offset error_1
call print
jmp execution_is_over
print_overlay_error2:
mov dx,offset error_2
call print
jmp execution_is_over
print_overlay_error3:
mov dx,offset error_3
call print
jmp execution_is_over
print_overlay_error4:
mov dx,offset error_4
call print
jmp execution_is_over
print_overlay_error5:
mov dx,offset error_5
call print
jmp execution_is_over
print_overlay_error8:
```

```
mov dx,offset error_8
call print
jmp execution_is_over
print_overlay_error10:
mov dx,offset error_10
call print
jmp execution_is_over
```

```
execution_is_over:
pop es
pop ds
pop dx
pop cx
pop bx
pop ax
ret
overlay_execute endp
```

```
main proc far
mov ax,data_
mov ds,ax
mov keep_psp,es
```

```
call free_memory
```

```
mov dx,offset overlay_name1
call find_path
call overlay_size
call overlay_execute
```

```
mov dx,offset overlay_name2
call find_path
call overlay_size
call overlay_execute
```

```
xor al,al
mov ah,4ch
int 21h
main endp
main_end_prt:
code_ ends
end main
```

Название файла: ovl1.asm

```
ovl_code_ segment
assume cs:ovl_code_, ds:nothing, es:nothing, ss:nothing
```

```
overlay proc far
push ax
push dx
push di
push ds
mov ax,cs
mov ds,ax
mov bx,offset address
add bx,21
mov di,bx
mov ax,cs
call wrd_to_hex
mov dx,offset address
```

```
call print
pop ds
pop di
pop dx
pop ax
retf
overlay 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
xor ah,ah
call byte_to_hex
mov [di],ah
dec di
mov [di],al
pop bx
ret
wrd_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
```

```
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
```

```
print proc near
push ax
mov ah,09h
int 21h
pop ax
ret
print endp
```

```
address db 'overlay1 address:      ',0dh,0ah,'$'
```

```
ovl_code_ ends
End
```


Название файла: ovl2.asm

ovl_code_ segment

assume cs:ovl_code_, ds:nothing, es:nothing, ss:nothing

overlay proc far

push ax

push dx

push di

push ds

mov ax,cs

mov ds,ax

mov bx,offset address

add bx,21

mov di,bx

mov ax,cs

call wrd_to_hex

mov dx,offset address

call print

pop ds

pop di

pop dx

pop ax

retf

overlay 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
xor ah,ah
call byte_to_hex
mov [di],ah
dec di
mov [di],al
pop bx
ret
wrd_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
```

```
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
```

```
print proc near
push ax
mov ah,09h
int 21h
pop ax
ret
print endp
```

```
address db 'overlay2 address:      ',0dh,0ah,'$'
```

```
ovl_code_ ends
end
```