

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

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

Студент гр. 8381

Киреев К.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

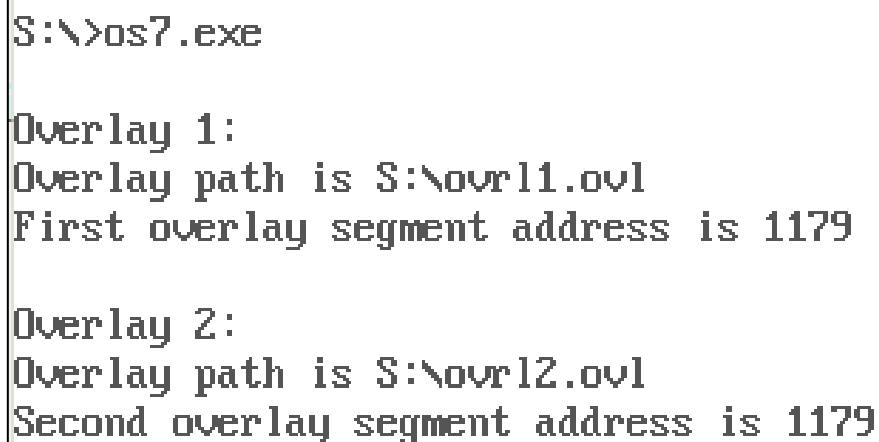
Исследование возможности построение загрузочного модуля оверлейной структуры. Исследуется структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Для запуска вызываемого оверлейного модуля используется функция 4B03h прерывания int 21h. Все загруженные и оверлейные модули находятся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

Выполнение работы.

Написан текст исходного EXE модуля, который выполняет следующие функции. Освобождает память для загрузки оверлеев. Читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки. Файл оверлейного сегмента загружается и выполняется. Освобождается память, отведенная для оверлейного сегмента. Затем предыдущие действия выполняются для следующего оверлейного сегмента. Также были написаны и отлажены оверлейные сегменты.

Полученный модуль был отлажен. Далее отлаженная программа была запущена. Результаты выполнения программы представлены на рис. 1.



```
S:\>os7.exe

Overlay 1:
Overlay path is S:\ovrl1.ovl
First overlay segment address is 1179

Overlay 2:
Overlay path is S:\ovrl2.ovl
Second overlay segment address is 1179
```

Рисунок 1 – Результат выполнения OS6.EXE.

Видно, что оверлеи запускаются с одного адреса.

Далее программа была запущена из другого каталога. Результаты выполнения представлены на рис. 2.

```
S:\ARCHIVE>os7

Overlay 1:
Overlay path is S:\ARCHIVE\ovrl1.ovl
First overlay segment address is 1179

Overlay 2:
Overlay path is S:\ARCHIVE\ovrl2.ovl
Second overlay segment address is 1179
```

Рисунок 2 – Результат выполнения OS6.EXE из другого каталога.

Далее отлаженная программа была запущена, когда одного оверлея не было в каталоге. Результаты выполнения программы представлены на рис. 3.

```
S:\ARCHIVE>os7

Overlay 1:
Overlay path is S:\ARCHIVE\ovrl1.ovl
First overlay segment address is 1179

Overlay 2:
Overlay path is S:\ARCHIVE\ovrl2.ovl

File not found
```

Рисунок 3 – Результат выполнения OS6.EXE, когда одного оверлея нет в каталоге.

Контрольные вопросы

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

При использовании в качестве оверлейного сегмента .COM модуля, необходимо вызывать его по смещению 100h, так как в .COM файлах код располагается с адреса 100h. В ином случае PSP не будет сформирован.

Вывод.

В процессе выполнения данной лабораторной работы была исследована возможность построения загрузочного модуля оверлейной структуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. OS7.ASM

Astack segment stack

dw 256 dup(?)

Astack ends

data segment

dos_4ah_err7 db 13,10,'Memory control block is
destroyed',13,10,'\$'

dos_4ah_err8 db 13,10,'Not enough memory to execute
function',13,10,'\$'

dos_4ah_err9 db 13,10,'Invalid memory block
address',13,10,'\$'

dos_4eh_err2 db 13,10,'File not found\$',13,10,'\$'

dos_4eh_err3 db 13,10,'Route not found\$',13,10,'\$'

dos_48h_err db 13,10,'Failed to allocate memory to load
overlay!',13,10,'\$'

dos_4b03h_err db 13,10,'Overlays did not load',13,10,'\$'

dos_4b03h_err1 db 13,10,'Function number is
invalid',13,10,'\$'

dos_4b03h_err2 db 13,10,'File not found',13,10,'\$'

dos_4b03h_err3 db 13,10,'File not found',13,10,'\$'

dos_4b03h_err4 db 13,10,'File not found',13,10,'\$'

dos_4b03h_err5 db 13,10,'Disk error',13,10,'\$'

dos_4b03h_err8 db 13,10,'Insufficient memory',13,10,'\$'

dos_4b03h_err10 db 13,10,'Wrong environment
string',13,10,'\$'

keep_psp dw 0

mesto db 16 dup (0)

overlays_path db 64 dup (0),'\$'

DTA db 43 DUP (?)

overlay_seg_address dw 0

overlay_prs dd 0

```
    ovr11 db 'ovr11.ovl',0
    ovr12 db 'ovr12.ovl',0
    msg_ovr11 db 13,10,'Overlay 1:',13,10,'$'
    msg_ovr12 db 13,10,'Overlay 2:',13,10,'$'
    msg_overlay_path db 'Overlay path is ', '$'
data ends
```

```
code segment
    assume cs:code, ds:data, ss:Astack
```

```
pushall macro
    irp case, <ax,bx,cx,dx,si,di,es,ds,sp,bp>
        push &case&
    endm
endm
popall macro
    irp case, <bp,sp,ds,es,di,si,dx,cx,bx,ax>
        pop &case&
    endm
endm
```

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

```
word_to_str proc near
    ;на входе ax число 16 бит
    ;si указатель на строку
    ;bx разрядность результата
    pushall
    cmp bx, 16
    ja end_wts
    cmp ax, 7FFFh
    jna plus
```

```

mov byte ptr [si], '-'
inc si
not ax
inc ax
plus:
xor cx, cx
jmp manipulation
manipulation:
xor dx, dx
div bx
mov di, ax
mov al, dl
cmp al, 10
sbb al, 69h
das
push di
lea di, mesto
add di, cx
mov byte ptr [di], al
pop di
mov ax, di
inc cx
test ax, ax
jz endrep
jmp manipulation
endrep:
lea di, mesto
add di, cx
copyrep:
dec di
mov dl, byte ptr [di]
mov byte ptr [si], dl
inc si
loop copyrep
end_wts:
popall
ret
word_to_str endp

```

```

prep proc near
    pushall
    mov bx, offset zseg
    mov ax, es
    sub bx, ax
    mov cl, 4
    shr bx, cl
    mov ah, 4Ah
    int 21h
    jnc exec_dos_4ah
    irpc case, 789
        cmp ax, &case&
        je type_dos_4ah_err&case&
    endm
    irpc case, 789
    type_dos_4ah_err&case&:
        lea dx, dos_4ah_err&case&
        call print
        mov ax, 4C00h
        int 21h
    endm
    exec_dos_4ah:
    popall
    ret
prep endp

```

```

prep_overlay_path proc
    push ds
    push dx
    mov dx, seg DTA
    mov ds, dx
    lea dx, DTA
    ;Dos Fn 1Ah Int 21h: позволяет определить адрес DTA для
    последующих операций
    ;ds:dx = адрес DTA
    mov ah, 1Ah
    int 21h

```



```

pop dx
pop ds
pushall
mov es, keep_psp
mov es, es:[2Ch]
xor si, si
lea di, overlays_path
skip_env:
    mov dl, es:[si]
    cmp dl, 00
    je end_env
    inc si
    jmp skip_env
end_env:
    inc si
    mov dl, es:[si]
    cmp dl, 00
    jne skip_env
    add si, 3
path_change:
    mov dl, es:[si]
    cmp dl, 00
    je final_name
    mov [di], dl
    inc si
    inc di
    jmp path_change
final_name:
mov si, bp
child_name:
    mov dl, byte ptr [si]
    mov byte ptr [di-7], dl
    inc di
    inc si
    test dl, dl
    jne child_name
popall
ret

```

```
prep_overlay_path endp
```

```
prep_overlay_size proc
```

```
    pushall
```

```
    xor cx, cx
```

```
    mov dx, seg overlays_path
```

```
    mov ds, dx
```

```
    lea dx, overlays_path
```

```
    ;Dos Fn 4Eh Int 21h: нахождение первого файла,  
соответствующего заданной спецификации
```

```
    ;cx=атрибуты искомых файлов
```

```
    ;ds:dx=адрес спецификации искомого файла
```

```
    mov ah,4Eh
```

```
    int 21h
```

```
    jnc exec_dos_4eh
```

```
irpc case, 23
```

```
    cmp ax, &case&
```

```
    je type_dos_4eh_err&case&
```

```
endm
```

```
irpc case, 23
```

```
type_dos_4eh_err&case&:
```

```
    lea dx, dos_4eh_err&case&
```

```
    call print
```

```
    mov ax, 4C00h
```

```
    int 21h
```

```
endm
```

```
exec_dos_4eh:
```

```
mov si, offset DTA
```

```
mov ax, [si+1Ah]
```

```
mov bx, [si+1Ch]
```

```
mov cl, 4
```

```
shr ax, cl
```

```
mov cl, 12
```

```
shl bx, cl
```

```
add ax, bx
```

```
add ax, 2
```

```
mov bx, ax
```

```
;Dos Fn 48h Int 21h: выделение блока памяти
```

```

;bx=требуемое число параграфов памяти
;на выходе: ax=сегментный адрес выделенного блока
mov ah, 48h
int 21h
jnc exec_dos_48h
lea dx, dos_48h_err
call print
mov ax, 4C00h
int 21h
exec_dos_48h:
mov overlay_seg_address, ax
popall
ret
prep_overlay_size endp

load_overlay proc
    pushall
    push ss
    push sp
    mov bx, seg overlay_seg_address
    mov es, bx
    lea bx, overlay_seg_address
    mov dx, seg overlays_path
    mov ds, dx
    lea dx, overlays_path
    ;Dos Fn 4803h Int 21h: загрузка оверлея в отведенную
    область памяти
    ;ds:dx=строка, содержащая путь к оверлею
    ;es:bx=указатель на блок параметров
    mov ax, 4B03h
    int 21h
    jnc exec_dos_4b03h
    lea dx, dos_4b03h_err
    call print
    irp case, <1,2,3,4,5,8,10>
        cmp ax, &case&
        je type_dos_4b03h_err&case&
    endm

```

```

    irp case, <1,2,3,4,5,8,10>
    type_dos_4b03h_err&case&:
        lea dx, dos_4b03h_err&case&
        call print
        jmp OVL_RET
    endm
exec_dos_4b03h:
    mov ax, overlay_seg_address
    mov word ptr overlay_prs+2, ax
    call overlay_prs
    mov es, ax
;Dos Fn 49h Int 21h: освобождение блока памяти
;es=сегментный адрес освобождаемого блока
    mov ax, 4900h
    int 21h
OVL_RET:
    mov es, keep_psp
    pop sp
    pop ss
    popall
    ret
load_overlay endp

main proc far
    push ds
    xor ax, ax
    push ax
    mov ax, DATA
    mov ds, ax
    mov keep_psp, es
    call prep
    lea dx, msg_ovr11
    call print
    lea bp, ovr11
    call prep_overlay_path
    lea dx, msg_overlay_path
    call print
    lea dx, overlays_path

```

```
    call print
    call prep_overlay_size
    call load_overlay
    lea dx, msg_ovr12
    call print
    lea bp, ovr12
    call prep_overlay_path
    lea dx, msg_overlay_path
    call print
    lea dx, overlays_path
    call print
    call prep_overlay_size
    call load_overlay
    mov ax, 4C00h
    int 21h
    ret
main endp
code ends
zseg segment
zseg ends
end main
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. OVRL1.ASM

```
ovrl1 segment
assume cs:ovrl1
main: jmp processing

info db 'First overlay segment address is      ',13,10,'$'
mesto db 16 dup (0)

processing proc far
    push ax
    push bx
    push dx
    push ds
    push si
    mov ax, cs
    mov ds, ax
    lea si, info+33
    mov bx, 16
    call word_to_str
    lea dx, info
    call print
    pop si
    pop ds
    pop dx
    pop bx
    pop ax
    retf
processing endp

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

```

word_to_str proc near
    ;на входе ax число 16 бит
    ;si указатель на строку
    ;bx разрядность результата
    push ax
    push bx
    push cx
    push dx
    push di
    push si
    cmp bx, 16
    ja end_wts
    cmp ax, 7FFFh
    jna plus
    mov byte ptr [si], '-'
    inc si
    not ax
    inc ax
plus:
    xor cx, cx
    jmp manipulation
manipulation:
    xor dx, dx
        div bx
        mov di, ax
        mov al, dl
        cmp al, 10
        sbb al, 69h
        das
        push di
        lea di, mesto
        add di, cx
        mov byte ptr [di], al
        pop di
        mov ax, di
    inc cx
    test ax, ax

```

```
jz endrep
jmp manipulation
endrep:
    lea di, mesto
        add di, cx
copyrep:
    dec di
    mov dl, byte ptr [di]
    mov byte ptr [si], dl
    inc si
    loop copyrep
end_wts:
pop si
pop di
pop dx
pop cx
pop bx
pop ax
ret
word_to_str endp
ovrl1 ends
end main
```


ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. OVRL2.ASM

```
ovrl2 segment
assume cs:ovrl2
main: jmp processing

info2 db 'Second overlay segment address is      ',13,10,'$'
mesto db 16 dup (0)

processing proc far
    push ax
    push bx
    push dx
    push ds
    push si
    mov ax, cs
    mov ds, ax
    lea si, info2+34
    mov bx, 16
    call word_to_str
    lea dx, info2
    call print
    pop si
    pop ds
    pop dx
    pop bx
    pop ax
    retf
processing endp

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

```

word_to_str proc near
    ;на входе ax число 16 бит
    ;si указатель на строку
    ;bx разрядность результата
    push ax
    push bx
    push cx
    push dx
    push di
    push si
    cmp bx, 16
    ja end_wts
    cmp ax, 7FFFh
    jna plus
    mov byte ptr [si], '-'
    inc si
    not ax
    inc ax
plus:
    xor cx, cx
    jmp manipulation
manipulation:
    xor dx, dx
        div bx
        mov di, ax
        mov al, dl
        cmp al, 10
        sbb al, 69h
        das
        push di
        lea di, mesto
        add di, cx
        mov byte ptr [di], al
        pop di
        mov ax, di
    inc cx
    test ax, ax

```

```
jz endrep
jmp manipulation
endrep:
    lea di, mesto
        add di, cx
copyrep:
    dec di
    mov dl, byte ptr [di]
    mov byte ptr [si], dl
    inc si
    loop copyrep
end_wts:
pop si
pop di
pop dx
pop cx
pop bx
pop ax
ret
word_to_str endp
ovrl2 ends
end main
```