

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

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

Студент гр. 8382

Нечепуренко Н.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

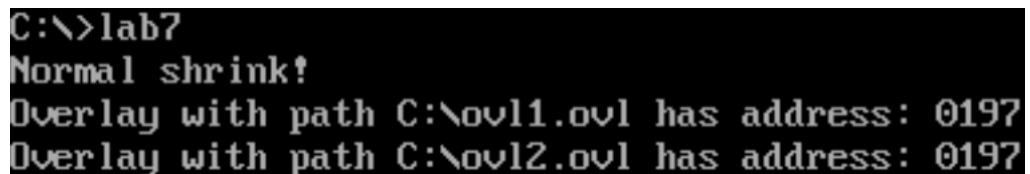
Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры. Исследуется структура оверлейного сегмента и способ загрузки и выполнения оверлейной структуры

Выполнение работы.

Для выполнения работы был реализован .exe модуль, который сначала освобождает неиспользуемую память, затем находит необходимые файлы оверлеев в текущей директории, аллоцирует память и загружает их. Оверлеи представляют собой модули, которые выводят на экран адрес начала их расположения в оперативной памяти.

Пусть в корневой директории находятся два оверлейных файла с названиями ovl1 и ovl2 соответственно (полный исходный код см. в Приложении Б), а также исполняемый модуль. Результат запуска программы приведён на рисунке 1.



```
C:\>lab7
Normal shrink!
Overlay with path C:\ovl1.ovl has address: 0197
Overlay with path C:\ovl2.ovl has address: 0197
```

Рисунок 1 – Корневая директория, присутствуют оба оверлея

Можно заметить, что оба оверлея были последовательно размещены по одному и тому же адресу.

Переместим файлы в директорию test и повторим запуск (см. рис. 2).



```
C:\TEST>lab7
Normal shrink!
Overlay with path C:\TEST\ovl1.ovl has address: 0197
Overlay with path C:\TEST\ovl2.ovl has address: 0197
```

Рисунок 2 – Директория test, присутствуют оба оверлея

Теперь удалим из директории второй оверлей, запустим программу ещё раз (см. рис. 3).

```
C:\TEST>del OVL2.OVL  
  
C:\TEST>lab?  
Normal shrink!  
Overlay with path C:\TEST\ovl1.ovl has address: 0197  
File not found!
```

Рисунок 3 – Директория test, второй оверлей удалён

Программа сообщила, что файл со вторым оверлеем не был найден. Удалим оставшийся оверлей, запустим программу (см. рис. 4).

```
C:\TEST>del OVL1.OVL  
  
C:\TEST>lab?  
Normal shrink!  
File not found!
```

Рисунок 4 – Директория test, нет оверлеев

Получаем ожидаемый вывод. Полный исходный код .exe модуля приведён в Приложении А.

Контрольные вопросы.

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

Оверлей представляет собой кодовый сегмент, функцию с точкой входа в 0, т.е. его вызов аналогичен дальнему колу, поэтому возврат из него происходит с помощью retf, а не функции 4ch прерывания int 21h, иначе вызывающая программа будет завершена. При использовании .com модулей стоит учитывать наличие с нулевого адреса PSP, а наличие целевой функции с адреса 100h, поэтому необходимо в вызывающей программе учесть смещение в 100h, т.е. размер префикса.

Выводы.

В ходе выполнения работы был получен загрузочный модуль оверлейной структуры. Были проанализированы требования к оверлеям, например, ограничения, которые накладываются при использовании .com модуля.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД МОДУЛЯ .EXE.

```
astack segment stack
    dw 100 dup(?)
astack ends

dataseg segment
    path db 64 dup(0), "$"
    dta db 43 dup(?)
    keep_psp dw 0
    name_idx dw 0
    overlay_name1 db "ovl1.ovl", 0, "$"
    overlay_name2 db "ovl2.ovl", 0, "$"
    overlay_seg dw 0
    call_addr dd 0
    msg_shrink_normal db "Normal shrink!", 13, 10, "$"
    msg_shrink_destroyed db "Control block is destroyed!", 13, 10, "$"
    msg_shrink_notenough db "Not enough memory!", 13, 10, "$"
    msg_shrink_invalidadr db "Invalid address!", 13, 10, "$"
    msg_shrink_error_offset dw offset msg_shrink_destroyed
                                dw offset msg_shrink_notenough
                                dw offset msg_shrink_invalidadr
    msg_alloc_mem_fnf db "File not found!", 13, 10, "$"
    msg_alloc_mem_pnf db "Path not found!", 13, 10, "$"
    msg_alloc_mem_error db "Memory allocation error!", 13, 10, "$"
    msg_load_nofunc db "Functions doesnt exist!", 13, 10, "$"
    msg_load_toomany db "Too many open files!", 13, 10, "$"
    msg_load_noaccess db "No access!", 13, 10, "$"
    msg_header db "Overlay with path ", "$"
dataseg ends

codeseg segment
    assume ds:dataseg, cs:codeseg, ss:astack

shrink_memory proc near
    push ax
    push bx
    push cx
    push di
    push si
    mov bx, offset code_end
    mov ax, es
    sub bx, ax ; get the amount of memory this program use
```

```

        mov cl, 4
        shr bx, cl ; bx / 16 -> in paragraphs
        mov ax, 4a00h
        int 21h
        jc shrink_memory_error_occured
shrink_memory_normal:
        mov di, offset msg_shrink_normal
        call print
        jmp shrink_memory_final
shrink_memory_error_occured:
        sub ax, 7
        shl ax, 1
        mov bx, ax
        mov si, offset msg_shrink_error_offset
        mov di, ds:[si+bx]
        call print
        mov ax, 4c00h
        int 21h
shrink_memory_final:
        pop si
        pop di
        pop cx
        pop bx
        pop ax
        ret
shrink_memory endp

set_dta proc near
        push ds
        push ax
        push dx

        mov dx, seg dta
        mov ds, dx
        mov dx, offset dta
        mov ah, 1ah
        int 21h

set_dta_final:
        pop dx
        pop ax
        pop ds

```

```

        ret
set_dta endp

construct_path proc near
    push es
    push si
    push di
    push ax
    push cx
    mov es, es:[2Ch] ; envir addr
    mov si, 0
construct_path_skip_envir:
    mov al, es:[si]
    cmp al, 0
    je construct_path_if_all_skipped
    inc si
    jmp construct_path_skip_envir

construct_path_if_all_skipped:
    inc si
    mov al, es:[si]
    cmp al, 0
    jne construct_path_skip_envir

construct_path_find_module_path:
    add si, 3
    mov di, offset path
construct_path_copy:
    mov al, es:[si]
    cmp al, 0
    je construct_path_copy_name
    mov [di], al
    inc di
    inc si
    jmp construct_path_copy

construct_path_copy_name:
    sub di, 8
    mov name_idx, di

construct_path_final:
    pop bx

```

```

        pop ax
        pop di
        pop si
        pop es
        ret
construct_path endp

set_name proc near
    push di
    push si
    push ax
    push cx
    mov di, name_idx
    mov si, ax
    mov cx, 10
set_name_loop:
    mov al, [si]
    mov [di], al
    inc si
    inc di
    loop set_name_loop
set_name_final:
    pop cx
    pop ax
    pop si
    pop di
    ret
set_name endp

alloc_mem proc near
    push cx
    push dx
    push ds
    push ax
    push es

    mov cx, 0
    mov dx, seg path
    mov ds, dx
    mov dx, offset path
    mov ah, 4eh
    int 21h

```



```

        jnc alloc_mem_success
        cmp ax, 2
        je alloc_mem_fnf
        cmp ax, 3
        je alloc_mem_pnf

alloc_mem_fnf:
        mov di, offset msg_alloc_mem_fnf
        jmp alloc_mem_found_error
alloc_mem_pnf:
        mov di, offset msg_alloc_mem_pnf
alloc_mem_found_error:
        call print
        pop es
        pop ax
        pop ds
        pop dx
        pop cx
        mov ax, 4c00h
        int 21h

alloc_mem_success:
        mov si, offset dta
        mov ax, [si+1Ah]
        mov bx, [si+1Ch]
        mov cl, 4
        shr ax, cl
        mov cl, 12
        shr bx, cl
        add bx, ax
        add bx, 2
alloc_mem_allocation:
        mov ah, 48h
        int 21h
        jc alloc_mem_error
alloc_mem_final:
        mov overlay_seg, ax
        pop es
        pop ax
        pop ds
        pop dx
        pop cx

```

```

        ret
alloc_mem_error:
        mov di, offset msg_alloc_mem_error
        call print
        pop es
        pop ax
        pop ds
        pop dx
        pop cx
        mov ax, 4c00h
alloc_mem endp

```

```

load_overlay proc near
        push bx
        push ax
        push dx
        push ds
        push ss
        push sp

        mov bx, seg overlay_seg
        mov es, bx
        mov bx, offset overlay_seg
        mov dx, seg path
        mov ds, dx
        mov dx, offset path
        mov ax, 4B03h
        int 21h
        jnc load_overlay_success
        cmp ax, 1
        je load_overlay_nofunc
        cmp ax, 2
        je load_overlay_nofile
        cmp ax, 3
        je load_overlay_nopath
        cmp ax, 4
        je load_overlay_toomany
        cmp ax, 5
        je load_overlay_noaccess
        cmp ax, 8
        je load_overlay_notenough

```

```

load_overlay_nofunc:
    mov di, offset msg_load_nofunc
    jmp load_overlay_fail

load_overlay_nofile:
    mov di, offset msg_alloc_mem_fnf
    jmp load_overlay_fail

load_overlay_nopath:
    mov di, offset msg_alloc_mem_pnf
    jmp load_overlay_fail

load_overlay_toomany:
    mov di, offset msg_load_toomany
    jmp load_overlay_fail

load_overlay_noaccess:
    mov di, offset msg_load_noaccess
    jmp load_overlay_fail

load_overlay_notenough:
    mov di, offset msg_shrink_notenough
    jmp load_overlay_fail

load_overlay_fail:
    call print
    jmp load_overlay_final

load_overlay_success:
    mov di, offset msg_header
    call print
    mov di, offset path
    call print

    mov ax, overlay_seg
    mov bx, offset call_addr
    mov [bx+2], ax
    call call_addr
    mov es, ax
    mov ax, 4900h
    int 21h

load_overlay_final:
    pop sp
    pop ss
    pop ds
    pop dx

```

```

        pop ax
        pop bx
        mov es, keep_psp
        ret
load_overlay endp

main proc near
    mov ax, dataseg
    mov ds, ax
    mov keep_psp, es
    call shrink_memory
    call set_dta
    call construct_path

    mov ax, offset overlay_name1
    call set_name
    call alloc_mem
    call load_overlay

    mov ax, offset overlay_name2
    call set_name
    call alloc_mem
    call load_overlay

    mov ax, 4c00h
    int 21h
main endp

print proc near
    ; prints di content
    push dx
    push ax
    mov ah, 9h
    mov dx, di
    int 21h
    pop ax
    pop dx
    ret
print endp
code_end:
codeseg ends
end main

```

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ОВЕРЛЕЕВ.

```
overlay segment
    assume cs:overlay, ds:overlay
    start: jmp main
    msg_address db "has address:      ", 13, 10, "$"

main proc near
    push ax
    push ds
    push di
    mov ax, cs
    mov ds, ax
    mov di, offset msg_address
    push di
    add di, 10h
    call WRD_TO_HEX
    pop di
    call print
    pop di
    pop ds
    pop ax
    retf
main endp

print proc near
    ; prints di content
    push dx
    push ax
    mov ah, 9h
    mov dx, di
    int 21h
    pop ax
    pop dx
    ret
print endp

WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
    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
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
;байт в AL переводится в два символа шест. числа в AX
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX ;в AL старшая цифра
    pop CX ;в AH младшая
    ret
BYTE_TO_HEX ENDP

overlay ends
end start

```