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. 9381 _____ Прибылов Н.А.

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 использованием полного пути.

Ход работы.

1) Написан и отлажен программный модуль типа .EXE, который выполняет функции:

- Проверяет, установлено ли пользовательское прерывание с вектором 09h.
- Если прерывание не установлено, то устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. Осуществляется выход по функции 4Ch прерывания int 21h.
- Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

2) Написаны и отлажены оверлейные сегменты, которые выводят адрес сегмента, в который они загружены.

3) Запущено отлаженное приложение. Оверлейные сегменты загружаются с одного и того же адреса, перекрывая друг друга.

4) Запущено приложение из другого каталога.

5) Запущено приложение в том случае, когда одного оверлея нет в каталоге.

6) Получившиеся результаты показаны в виде скриншотов.

Результат работы программы.

```
X:\>lab7.exe
Memory has been freed
Memory allocation successful
Load successful

file_ovl address: 01F2

Memory allocation successful
Load successful

file_ovl address: 01F2

X:\>_
```

Программа запущена

```
X:\LAB7>lab7.exe
Memory has been freed
Memory allocation successful
Load successful

file_ovl address: 01F2

Memory allocation successful
Load successful

file_ovl address: 01F2

X:\LAB7>_
```

Программа запущена из другого каталога

Программа запущена, первый оверлей в другой директории

Программа запущена, оба оверлея в другой директории

Контрольные вопросы.

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

В COM-модуле необходимо поместить значение регистра CS в регистр DS после записи значений в стек (адрес сегмента данных = адрес сегмента кода). Также при обращении к оверлейному сегменту необходимо обращаться к сегменту, смещённому на 100h.

Выводы.

Была исследована структура, способ загрузки и выполнение оверлейных сегментов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab7.asm

```
data segment
     file1      db "ovl1.ovl", 0
     file2      db "ovl2.ovl", 0
     program    dw 0
     dta_mem    db 43 dup(0)
     mem_flag   db 0
     cl_pos     db 128 dup(0)
     ovls_addr  dd 0
     keep_psp   dw 0

     eof                    db 0dh, 0ah, '$'
     str_free_memory         db 'Memory has been freed' , 0dh, 0ah,
'$'
     str_file_error         db 'Error: file not found, load error',
0dh, 0ah, '$'
     str_route_error            db 'Error: route not found, load
error', 0dh, 0ah, '$'
     str_normal_end         db 'Load successful', 0dh, 0ah, '$'
     str_allocation_mem_end db 'Memory allocation successful', 0dh,
0ah, '$'
     str_all_file_error        db 'Error: file not found, memory
allocation error' , 0dh, 0ah, '$'
     str_all_route_error      db 'Error: route not found, memory
allocation error' , 0dh, 0ah, '$'
     end_data               db 0
data ends

stacks segment stack
     dw 128 dup(?)
stacks ends

code segment

assume cs:code, ds:data, ss:stacks

print_str proc
     push ax
     mov ah, 09h
     int 21h
     pop ax
     ret
print_str endp

free_memory proc
     push ax
     push bx
     push cx
     push dx

     mov ax, offset end_data
     mov bx, offset exit
```

```
      add bx, ax

      mov cl, 4
      shr bx, cl
      add bx, 2bh
      mov ah, 4ah
      int 21h

free_mem_end:
      mov mem_flag, 1
      mov dx, offset str_free_memory
      call print_str

end_free:
      pop dx
      pop cx
      pop bx
      pop ax
      ret
free_memory endp

load_proc proc
      push ax
      push bx
      push cx
      push dx
      push ds
      push es

      mov ax, data
      mov es, ax
      mov bx, offset ovls_addr
      mov dx, offset cl_pos
      mov ax, 4b03h
      int 21h

      jnc _loads

file_err:
      cmp ax, 2
      jne route_err
      mov dx, offset str_file_error
      call print_str
      jmp _loade
route_err:
      cmp ax, 3
      jne _loads
      mov dx, offset eof
      call print_str
      mov dx, offset str_route_error
      call print_str
      jmp _loade
_loads:
      mov dx, offset str_normal_end
      call print_str

      mov ax, word ptr ovls_addr
      mov es, ax
```

```
      mov word ptr ovls_addr, 0
      mov word ptr ovls_addr+2, ax

      call ovls_addr
      mov es, ax
      mov ah, 49h
      int 21h

_loade:
      pop es
      pop ds
      pop dx
      pop cx
      pop bx
      pop ax
      ret
load_proc endp

path proc
      push ax
      push bx
      push cx
      push dx
      push di
      push si
      push es

      mov program, dx

      mov ax, keep_psp
      mov es, ax
      mov es, es:[2ch]
      mov bx, 0

findz:
      inc bx
      cmp byte ptr es:[bx-1], 0
      jne findz

      cmp byte ptr es:[bx+1], 0
      jne findz

      add bx, 2
      mov di, 0

_loop:
      mov dl, es:[bx]
      mov byte ptr [cl_pos+di], dl
      inc di
      inc bx
      cmp dl, 0
      je _end_loop
      cmp dl, '\'
      jne _loop
      mov cx, di
      jmp _loop
_end_loop:
      mov di, cx
```

```
      mov si, program

_fn:
      mov dl, byte ptr [si]
      mov byte ptr [cl_pos+di], dl
      inc di
      inc si
      cmp dl, 0
      jne _fn


      pop es
      pop si
      pop di
      pop dx
      pop cx
      pop bx
      pop ax
      ret
path endp

allocation_mem proc
      push ax
      push bx
      push cx
      push dx

      push dx
      mov dx, offset dta_mem
      mov ah, 1ah
      int 21h
      pop dx
      mov cx, 0
      mov ah, 4eh
      int 21h

      jnc _all_success

_allfile_err:
      cmp ax, 2
      je _allroute_err
      mov dx, offset str_all_file_error
      call print_str
      jmp _all_end
_allroute_err:
      cmp ax, 3
      mov dx, offset str_all_route_error
      call print_str
      jmp _all_end

_all_success:
      push di
      mov di, offset dta_mem
      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 ovls_addr, ax
      mov dx, offset str_allocation_mem_end
      call print_str

_all_end:
      pop dx
      pop cx
      pop bx
      pop ax
      ret
allocation_mem endp

begin_ovl proc
      push dx
      call path
      mov dx, offset cl_pos
      call allocation_mem
      call load_proc
      pop dx
      ret
begin_ovl endp

begin proc far
      push ds
      xor ax, ax
      push ax
      mov ax, data
      mov ds, ax
      mov keep_psp, es
      call free_memory
      cmp mem_flag, 0
      je _end

      mov dx, offset file1
      call begin_ovl
      mov dx, offset eof
      call print_str
      mov dx, offset file2
      call begin_ovl

_end:
      xor al, al
      mov ah, 4ch
      int 21h

begin endp
exit:
code ends
end begin
```