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 _____ Терехов А.Е.

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 оверлейных сегментов.

Ход работы.

1. Был написан и отлажен EXE-модуль. На рисунке 1 представлен результат выполнения программы при условии, что оба модуля находятся в одном каталоге.

```
C:\>LR7.EXE
C:\OVL1.OVL                                    OVERLAY1: 1179
C:\OVL2.OVL                                    OVERLAY2: 1179
```

Рисунок 1. Результат выполнения программы.

Результаты запуска файла, находящихся в каталоге "7" представлены на рисунке 2.

```
C:\>7\LR7.EXE
C:\7\OVL1.OVL                                  OVERLAY1: 1179
C:\7\OVL2.OVL                                  OVERLAY2: 1179
```

Рисунок 2. Результат выполнения программы, находящейся в каталоге "7".

В случае если модули находятся в разных каталогах, вызывающая программа не сможет найти вызываемый оверлей (рисунки 3, 4).

```
C:\>LR7.EXE
Файл не найден.
```

Рисунок 3. Результат выполнения программы, в случае, когда первый оверлей находятся в другом каталоге.

```
C:\>LR7.EXE
C:\OVL1.OVL                                    OVERLAY1: 1179
Файл не найден.
```

Рисунок 4. Результат выполнения программы, в случае, когда второй оверлей находятся в другом каталоге.

Ответы на вопросы.

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

Чтобы использовать .com модули необходимо учесть смещение на размер префикса в 100h свойственное .com программам. Также оверлей это лишь функция с точкой входа 0. Вызов оверлея равносилен вызову Far функции, поэтому возврат из нее происходит по retf а не по 4ch прерывания dos.

Вывод.

В ходе работы был построен загрузочный модуль оверлейной структуры, то есть программа, которая в ходе работы вызывает функцию, находящуюся в оверлейном модуле. Были получены навыки написания оверлеев.

ПРИЛОЖЕНИЕ А

```
AStack SEGMENT STACK
    DW 100h DUP(0)
AStack ENDS
DATA SEGMENT
    M_DESTROYED_MCB db'Разрушен управляющий блок памяти',0Dh,0Ah,'$'
    M_NO_MEMORY     db'Недостаточно памяти для выполнения функции',
0Dh,0Ah,'$'
    M_WRONG_ADRESS  db'Неверный адрес блока памяти',0Dh,0Ah,'$'
    KEEP_PSP dw 0
    PATH db 64 dup(0),'$'
    DTA db 43 dup(0)
    NAME_OVL1 db 'OVL1.OVL', 0
    NAME_OVL2 db 'OVL2.OVL', 0
    OVL_SEGMENT dw 0
    OVL_ADD dd 0
    M_PATH_NOT_EXIST    db'Путь не найден.',0Dh,0Ah,'$'
    M_FILE_NOT_EXIST    db'Файл не найден.',0Dh,0Ah,'$'
    M_DISK_ERROR        db'Ошибка диска.',0Dh,0Ah,'$'
    M_NOT_ENOUGH_MEM    db'Недостаточно памяти.',0Dh,0Ah,'$'
    M_MEMORY_ERR        db'Ошибка памяти',0Dh,0Ah,'$'
    M_L_NO_FUN          db'Несуществующая функция',0Dh,0Ah,'$'
    M_L_SO_MANY_FILES   db'Слишком много открытых файлов',0Dh,0Ah,'$'
    M_L_NO_ACCESS       db'Нет доступа',0Dh,0Ah,'$'

DATA ENDS

CODE SEGMENT
ASSUME ss: AStack, ds:DATA, cs:CODE

WRITE proc near
    push ax
    mov ah,09h
    int 21h
    pop ax
    ret
WRITE ENDP

FREE_MEM proc near
    mov bx, offset LAST
    mov ax, es
    sub bx, ax
    mov cl, 4
    shr bx, cl
    mov ah, 4ah
    int 21h
    jc ERROR_FREE
    jmp FREE_OK
ERROR_FREE:
    cmp ax, 7
    je DESTR_MCB
    cmp ax, 8
    je NO_MEM
    cmp ax, 9
    je WRONG_ADRESS
DESTR_MCB:
```

```
    lea dx, M_DESTROYED_MCB
    jmp PRINT_ERROR
NO_MEM:
    lea dx, M_NO_MEMORY
    jmp PRINT_ERROR
WRONG_ADRESS:
    lea dx, M_WRONG_ADRESS
    jmp PRINT_ERROR
PRINT_ERROR:
    call WRITE
    xor ax, ax
    mov ah, 4ch
    int 21h
FREE_OK:
    ret
FREE_MEM ENDP

SET_DTA proc near
    push dx
    push ds
    mov dx, seg DTA
    mov ds, dx
    lea dx, DTA
    mov ah, 1Ah
    int 21h
    pop ds
    pop dx
    ret
SET_DTA ENDP

SET_PATH proc near
    push es
    push si
    push di
    push ax
    push cx
    mov es, es:[2ch]
    lea di, PATH
    mov si, 0
cycle_env:
    mov al, es:[si]
    cmp al, 0
    je cycle_env_end
    inc si
    jmp cycle_env
cycle_env_end:
    inc si
    mov al, es:[si]
    cmp al, 0
    jne cycle_env
    add si, 3
cycle_path:
    mov al, es:[si]
    cmp al, 0
    je path_end
    mov [di], al
    inc si
```

```
    inc di
    jmp cycle_path
path_end:
    mov si, bp
new_path:
    mov al, byte ptr [si]
    mov byte ptr [di-7], al
    inc di
    inc si
    cmp al, 0
    jne new_path
    mov dl, '$'
    mov byte ptr [di-7], al

    pop cx
    pop ax
    pop di
    pop si
    pop es
    ret
SET_PATH ENDP

_SIZE proc near
    push ds
    push es
    xor cx, cx
    mov ax, seg PATH
    mov ds, ax
    lea dx, PATH
    mov ah, 4Eh
    xor al, al
    int 21h
    jnc file_here
    cmp ax, 2
    je file_not_exist
    cmp ax, 3
    je path_not_exist
file_not_exist:
    lea dx, M_FILE_NOT_EXIST
    jmp error_size
path_not_exist:
    lea dx, M_PATH_NOT_EXIST
    jmp error_size
error_size:
    call WRITE
    pop es
    pop ds
    xor al, al
    mov ah, 4ch
    int 21h
file_here:
    lea bx, DTA
    mov  ax, [bx+1ch]
     mov  bx, [bx+1ah]
     mov  cl, 12
     shl  ax, cl
     mov  cl, 4
```

```
      shr  bx, cl
      add  bx, ax
      inc  bx
      mov ah,48h
      int 21h
    jc mem_err
    mov OVL_SEGMENT, ax
    pop es
    pop ds
    ret
mem_err:
    lea dx, M_MEMORY_ERR
    call WRITE
    pop es
    pop ds
    xor al, al
    mov ah, 4ch
    int 21h
_SIZE ENDP

LOAD proc near
    push ds
    push ss
    push sp
    push es
    push ds
    mov bx, seg OVL_SEGMENT
    mov es, bx
    lea bx, OVL_SEGMENT
    mov dx, seg PATH
    mov ds, dx
    lea dx, PATH
    mov ah, 4Bh
    mov al, 3h
    int 21h
    pop ds
    jnc load_ok
    cmp ax, 1
    je fun_not_exist
    cmp ax, 2
    je file_not_found
    cmp ax, 3
    je path_not_found
    cmp ax, 4
    je so_many_open
    cmp ax, 5
    je no_access
    cmp ax, 8
    je not_enough_mem
fun_not_exist:
    lea dx, M_L_NO_FUN
    jmp not_load
file_not_found:
    lea dx, M_FILE_NOT_EXIST
    jmp not_load
path_not_found:
    lea dx, M_PATH_NOT_EXIST
```

```
    jmp not_load
so_many_open:
    lea dx, M_L_SO_MANY_FILES
    jmp not_load
no_access:
    lea dx, M_L_NO_ACCESS
    jmp not_load
not_enough_mem:
    lea dx, M_NOT_ENOUGH_MEM
    jmp not_load
not_load:
    call WRITE
    jmp end_load
load_ok:
    lea dx, PATH
    call WRITE
    mov ax, DATA
    mov ds, ax
    mov ax, OVL_SEGMENT

    mov word ptr OVL_ADD+2, ax
    call OVL_ADD
    mov ax, OVL_SEGMENT
    mov es, ax
    xor al, al
    mov ah, 49h
    int 21h
end_load:
    pop es
    pop sp
    pop ss
    pop ds
    ret
LOAD ENDP

MAIN proc far
    mov ax, DATA
    mov ds, ax

    mov KEEP_PSP, es
    call FREE_MEM
    call SET_DTA
    lea bp, NAME_OVL1
    call SET_PATH
    call _SIZE
    call LOAD

    lea bp, NAME_OVL2
    call SET_PATH
    call _SIZE
    call LOAD
    xor ax, ax
    mov ah, 4ch
    int 21h
    ret
MAIN ENDP
CODE ENDS
```

```
LAST SEGMENT
LAST ENDS
END MAIN
```