

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

Муковский Д.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

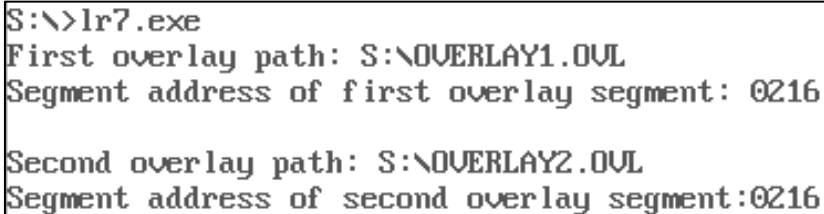
Исследование возможности построение загрузочного модуля оверлейной структуры. Исследуется структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Для запуска вызываемого оверлейного модуля используется функция 4B03h прерывания int 21h. Все загруженные и оверлейные модули находятся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

Результат отлаженной программы, когда оверлеи находятся в текущей директории представлен на рис.1.



```
S:\>lr7.exe
First overlay path: S:\OVERLAY1.OVL
Segment address of first overlay segment: 0216

Second overlay path: S:\OVERLAY2.OVL
Segment address of second overlay segment:0216
```

Рисунок 1 – Результат выполнения LR7.EXE

Далее программа была запущена из другой директории. Результаты выполнения представлены на рис. 2.

```
S:\TEST>lr7.exe
First overlay path: S:\TEST\OVERLAY1.OVL
Segment address of first overlay segment: 0216

Second overlay path: S:\TEST\OVERLAY2.OVL
Segment address of second overlay segment:0216
```

Рисунок 2 – Результат выполнения LR7.EXE из другой директории

Результат запуска отлаженной программы, когда только второй оверлей находится с исполняемым файлом в одном каталоге, представлен на рис.3.

```
S:\>lr7.exe
First overlay path:
File not found

Second overlay path: S:\OVERLAY2.OVL
Segment address of second overlay segment:0216
```

Рисунок 3 – Результат выполнения LR7.EXE с только вторым оверлеем

Результат запуска отлаженной программы, когда только первый оверлей находится с исполняемым файлом в одном каталоге, представлен на рис.4.

```
S:\>lr7.exe
First overlay path: S:\OVERLAY1.OVL
Segment address of first overlay segment: 0216

Second overlay path:
File not found
```

Рисунок 4 – Результат выполнения LR7.EXE с только первым оверлеем

Результат запуска отлаженной программы, когда ни один из оверлеев не находится в директории с исполняемым файлом, представлен на рис.5.

```
S:\>lr7.exe
First overlay path:
File not found

Second overlay path:
File not found
```

Рисунок 5 – Результат выполнения LR7.EXE, когда первого и второго оверлея нет в каталоге.

Контрольные вопросы

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

В случае использования .COM модуля в качестве оверлейного сегмента, необходимо вызывать его по смещению 100h, так как в данных модулях код располагается с адреса 100h. Если этим пренебречь PSP не будет сформирован.

Вывод.

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

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. LR7.ASM

DATA SEGMENT

OVL1_NAME db 'OVERLAY1.OVL', 0
OVL2_NAME db 'OVERLAY2.OVL', 0

OVL1_INFO db 'First overlay path: \$'
OVL2_INFO db 13,10,'Second overlay path: \$'

SAVE_PSP dw 0

ERR7_MEM db 13,10,'Memory control block is destroyed',13, 10,'\$'
ERR8_MEM db 13,10,'Not enough memory for function',13, 10,'\$'
ERR9_MEM db 13,10,'Invalid adress',13, 10,'\$'

ERR2_SIZE db 13,10,'File not found',13, 10,'\$'
ERR3_SIZE db 13,10,'Route not found',13, 10,'\$'

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

ERR1_LOAD db 13,10,'Incorrect function number',13, 10,'\$'
ERR2_LOAD db 13,10,'File not found',13, 10,'\$'
ERR3_LOAD db 13,10,'Route not found',13, 10,'\$'
ERR4_LOAD db 13,10,'Too many opened files',13, 10,'\$'
ERR5_LOAD db 13,10,'Disk error',13, 10,'\$'
ERR8_LOAD db 13,10,'Not enough memory',13, 10,'\$'
ERRA_LOAD db 13,10,'Invalid environment',13, 10,'\$'

CHECK_OVL db 0
DTA db 43 DUP (?)
OVERLAY_PATH db 50 dup (0),'\$'
OVERLAY_SEG_ADR dw 0
OVERLAY_A dd 0
DATA_END db 0

DATA ENDS

ASTACK SEGMENT STACK
DW 100 DUP(?)
ASTACK ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA, SS:ASTACK

PRINT PROC NEAR
 push ax
 mov ah, 09h
 int 21h
 pop ax
 ret
PRINT ENDP

FREE_MEMORY PROC NEAR
 push ax

```

    push bx
    push cx
    push dx
    push es
    push ds
    push si
    push di
    push ss
    push sp

    mov BX, offset PROG_END
    mov AX, offset DATA_END
    add BX, AX
    add BX, 40Fh
    mov CL, 4
    shr BX, CL
    mov AX, 4A00h
    int 21h

    jnc  END_FUNC_FM

    irpc case, 789
        cmp ax, &case&
        je ERRM_&case&
    endm

    irpc met, 789
        ERRM_&met&:
        mov dx, offset ERR&met&_MEM
        call PRINT
        mov ax, 4C00h
        int 21h
    endm

    END_FUNC_FM:

    pop sp
    pop ss
    pop di
    pop si
    pop ds
    pop es
    pop dx
    pop cx
    pop bx
    pop ax

    RET
FREE_MEMORY      ENDP

GET_OVL_PATH PROC NEAR
    push ax
    push bx
    push cx
    push dx
    push es
    push ds
    push si

```

```

        push di
        push ss
        push sp

        mov es, SAVE_PSP
        mov es, es:[2Ch]
        xor si,si

        mov di, offset OVERLAY_PATH
SKIP_ENVIR:
        mov dl, es:[si]
        cmp dl, 00
        je SKIP_ENVIR2
        inc si
        jmp SKIP_ENVIR

SKIP_ENVIR2      :
        inc si
        mov dl, es:[si]
        cmp dl, 00
        jne SKIP_ENVIR
        add si, 3

GET_PATH:
        mov dl, es:[si]
        cmp dl, 00
        je REWRITE_NAME
        mov [di], dl
        inc si
        inc di
        jmp GET_PATH

REWRITE_NAME:
        mov si,bp ;в bp имя ovl
FILE_NAME:
        mov dl, byte ptr [si]
        mov byte ptr [di-7], dl
        inc di
        inc si
        test dl, dl
        jne FILE_NAME

        pop sp
        pop ss
        pop di
        pop si
        pop ds
        pop es
        pop dx
        pop cx
        pop bx
        pop ax
        ret
GET_OVL_PATH ENDP

SIZE_OF_OVL      PROC NEAR
        push ax
        push bx

```

```
push cx
push dx
push es
push ds
push si
push di
push ss
push sp
```

```
mov dx, seg DTA
mov ds, dx
lea dx, DTA
mov ah, 1Ah
int 21h
```

```
mov dx, seg OVERLAY_PATH
mov ds, dx
mov ah, 4Eh
xor cx, cx
mov dx, offset OVERLAY_PATH
int 21h
```

```
jnc NO_SIZE_ERR
```

```
irpc case, 23
    cmp ax, &case&
    je ERRS_&case&
endm
```

```
irpc met, 23
    ERRS_&met&:
        mov dx, offset ERR&met&_SIZE
        call PRINT
        mov CHECK_OVL, 1
        jmp END_SIZE
endm
```

NO_SIZE_ERR :

```
mov si, offset DTA
```

```
mov bx, [si+1Ch]
mov cl, 12
shr bx, cl
```

```
mov ax, [si+1Ah]
mov cl, 4
shr ax, cl
```

```
add bx, ax
add bx, 2
```

```
mov ax, 4800h
int 21h
jnc NO_ERROR_LOAD_OVL
```



```

    mov dx, offset ERR_LOAD_OVERLAY
    call PRINT
    mov ax, 4C00h
    int 21h

NO_ERROR_LOAD_OVL:

    mov OVERLAY_SEG_ADR, ax
END_SIZE:
    pop sp
    pop ss
    pop di
    pop si
    pop ds
    pop es
    pop dx
    pop cx
    pop bx
    pop ax
    ret
SIZE_OF_OVL ENDP

LOAD_OVL PROC NEAR
    push ax
    push bx
    push cx
    push dx
    push es
    push ds
    push si
    push di
    push ss
    push sp

    push ss
    push sp

    mov dx, offset OVERLAY_PATH
    push ds
    pop es
    mov bx, offset OVERLAY_SEG_ADR

    mov ax, 4B03h
    int 21h
    jnc NO_ERROR_LOAD

    irpc case, 123458A
        cmp ax, 0&case&h
        je ERRL_&case&
    endm

    irpc met, 123458A
        ERRL_&met&:
            mov dx, offset ERR&met&_LOAD
            call PRINT

```

```

                                jmp END_LOAD_OVL
endm

NO_ERROR_LOAD:
mov ax, OVERLAY_SEG_ADR
mov word ptr OVERLAY_A+2, ax
call OVERLAY_A
mov es, ax
mov ax, 4900h
int 21h

END_LOAD_OVL:

mov es, SAVE_PSP

pop sp
pop ss

pop sp
pop ss
pop di
pop si
pop ds
pop es
pop dx
pop cx
pop bx
pop ax
ret
LOAD_OVL ENDP

MAIN PROC FAR
push ds
xor ax, ax
push ax
mov ax, DATA
mov ds, ax
mov SAVE_PSP, es
call FREE_MEMORY

;first ovl
mov dx, offset OVL1_INFO
call PRINT

mov bp, offset OVL1_NAME
call GET_OVL_PATH

call SIZE_OF_OVL

cmp CHECK_OVL, 1
je SECOND

mov dx, offset OVERLAY_PATH
call PRINT

```

```

        call LOAD_OVL

        ;second ovl
SECOND:
        mov CHECK_OVL,0
        mov dx, offset OVL2_INFO
        call PRINT

        mov bp,offset OVL2_NAME
        call GET_OVL_PATH

        call SIZE_OF_OVL
        cmp CHECK_OVL,1
        je END_PROG

        mov dx, offset OVERLAY_PATH
        call PRINT

        call LOAD_OVL

END_PROG:
        xor al, al
        mov ah, 4Ch
        int 21H
        ret

MAIN ENDP
PROG_END:
CODE ENDS
END MAIN

```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. OVERLAY1.ASM

```
OVERLAY_1 SEGMENT
ASSUME CS:OVERLAY_1, DS:NOTHING, ES:NOTHING, SS:NOTHING

MAIN:JMP BEGIN

SEG_ADR db      13,10,'Segment address of first overlay segment:
', 13, 10, '$'
```

```
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 цифра
                RET
BYTE_TO_HEX     ENDP
```

```
;-----
-----
```

```
WRD_TO_HEX PROC NEAR
;перевод в 16 с/с 16 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
                XOR  AH, AH
                CALL BYTE_TO_HEX
                MOV  [DI], AH
                DEC  DI
                MOV  [DI], AL
                POP  BX
```

```
RET
WRD_TO_HEX ENDP
```

```
;-----
-----
```

```
PRINT PROC NEAR
    PUSH AX
    MOV AH, 09H
    INT 21H
    POP AX
    RET
PRINT ENDP
```

```
;-----
-----
```

```
BEGIN PROC FAR
    push ax
    push dx
    push di
    push ds

    mov     ax, cs
    mov     ds, ax
    mov     bx, offset SEG_ADR
    add     bx, 47
    mov     di, bx
    mov     ax, cs
    call    WRD_TO_HEX
    mov     dx, offset SEG_ADR
    call    PRINT

    pop     ds
    pop     di
    pop     dx
    pop     ax
    retf
BEGIN ENDP
```

```
OVERLAY_1 ENDS
END
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. OVERLAY2.ASM

```
OVERLAY_2 SEGMENT
ASSUME CS:OVERLAY_2, DS:NOTHING, ES:NOTHING, SS:NOTHING

MAIN:JMP BEGIN

SEG_ADR db      13,10,'Segment address of second overlay segment:  ',
13, 10, '$'

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 цифра
                RET
BYTE_TO_HEX     ENDP

;-----
-----

WRD_TO_HEX PROC NEAR
;перевод в 16 с/с 16 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
        XOR     AH, AH
        CALL    BYTE_TO_HEX
        MOV     [DI], AH
        DEC     DI
        MOV     [DI], AL
        POP     BX
        RET
WRD_TO_HEX ENDP

```

```

;-----
-----

```

```

PRINT PROC NEAR
        PUSH    AX
        MOV     AH, 09H
        INT     21H
        POP     AX
        RET
PRINT ENDP

```

```

;-----
-----

```

```

BEGIN PROC FAR
        push    ax
        push    dx
        push    di
        push    ds

        mov     ax, cs
        mov     ds, ax
        mov     bx, offset SEG_ADR
        add     bx, 47
        mov     di, bx
        mov     ax, cs
        call    WRD_TO_HEX
        mov     dx, offset SEG_ADR
        call    PRINT

        pop     ds
        pop     di
        pop     dx
        pop     ax
        retf
BEGIN ENDP

```

```

OVERLAY_2 ENDS
END

```