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 _____ Любимов В.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 с использованием полного пути.

Описание функций и структуры данных

1. PRINT_MES – при помощи функции 9h из прерывания 21h выводит строку на экран.
2. FREE_MEM_FOR_OVER – освобождает память для запуска вызываемого оверлейного модуля. Обрабатывает возможные ошибки.
3. LOAD_OVERLAY – загружает выполняемый оверлейный модуль. Обрабатывает возможные ошибки.
4. PATH_MAKING – создаёт путь для выполняемого оверлейного модуля.
5. ALLOC_MEMORY – выделяет память под оверлейный модуль.

Ход выполнения работы

1. Запускаем программу из каталога с разработанными модулями.

```
F:\>lb7
Memory for calling module has been cleared!
Memory has been successfully allocated!
Loaded has been successfully finished!

Address of the first overlay module:  0201

Memory has been successfully allocated!
Loaded has been successfully finished!

Address of the second overlay module:  0201
```

2. Запускаем программу из другого каталога.

```
F:\TEST>lb7
Memory for calling module has been cleared!
Memory has been successfully allocated!
Loaded has been successfully finished!

Address of the first overlay module:  0201

Memory has been successfully allocated!
Loaded has been successfully finished!

Address of the second overlay module:  0201
```

3. Первый оверлейный модуль находится в другом каталоге.

```
F:\TEST>lb7
Memory for calling module has been cleared!
File has not been found!
File has not been found!

Memory has been successfully allocated!
Loaded has been successfully finished!

Address of the second overlay module:  0201
```

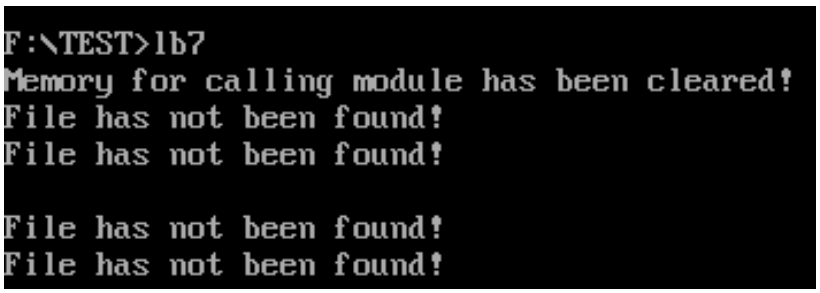
4. Второй оверлейный модуль находится в другом каталоге.

```
F:\TEST>lb7
Memory for calling module has been cleared!
Memory has been successfully allocated!
Loaded has been successfully finished!

Address of the first overlay module:  0201

File has not been found!
File has not been found!
```

5. Оба оверлейных модуля находятся в другом каталоге.



Ответы на контрольные вопросы

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

Необходимо вызвать такой модуль со смещением 100h (с смещения в .COM модулях располагается код), разместив в начале выделенной памяти PSP. Это нужно для корректного формирования PSP.

Вывод

В ходе выполнения работы была изучена возможность построения модуля оверлейной структуры.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

over_1.asm

```
CODE SEGMENT
	ASSUME CS:CODE, DS:nothing, SS:nothing

	MAIN_1 PROC FAR
		push ax
		push dx
		push ds
		push di

		mov ax, cs
		mov ds, ax
		mov di, offset ovl_addres
		add di, 43
		call WORD_TO_HEX
		mov dx, offset ovl_addres
		call PRINT_MES

		pop di
		pop ds
		pop dx
		pop ax
		retf
	MAIN_1 ENDP

	ovl_addres db 13, 10, "Address of the first overlay module:          ", 13, 10,
'$'

	PRINT_MES PROC
		push dx
		push ax

		mov ah, 09h
		int 21h

		pop ax
		pop dx
		ret
	PRINT_MES ENDP


	TETR_TO_HEX PROC
		and al,0fh
		cmp al,09
		jbe next
		add al,07
	next:
		add al,30h
		ret
	TETR_TO_HEX ENDP


	BYTE_TO_HEX PROC
		push cx
		mov ah, al
		call TETR_TO_HEX
		xchg al,ah
		mov cl,4
		shr al,cl
```

```
        call TETR_TO_HEX
        pop cx
        ret
    BYTE_TO_HEX ENDP


    WORD_TO_HEX PROC
        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
    WORD_TO_HEX ENDP

CODE ENDS
     END MAIN_1
```

over_2.asm

```
CODE SEGMENT
    ASSUME CS:CODE, DS:nothing, SS:nothing

    MAIN_1 PROC FAR
        push ax
        push dx
        push ds
        push di

        mov ax, cs
        mov ds, ax
        mov di, offset ovl_addres
        add di, 44
        call WORD_TO_HEX
        mov dx, offset ovl_addres
        call PRINT_MES

        pop di
        pop ds
        pop dx
        pop ax
        retf
    MAIN_1 ENDP

    ovl_addres db 13, 10, "Address of the second overlay module:          ", 13, 10,
'$'

    PRINT_MES PROC
        push dx
        push ax

        mov ah, 09h
        int 21h

        pop ax
        pop dx
```

```
        ret
    PRINT_MES ENDP


    TETR_TO_HEX PROC
        and al,0fh
        cmp al,09
        jbe next
        add al,07
    next:
        add al,30h
        ret
    TETR_TO_HEX ENDP


    BYTE_TO_HEX PROC
        push cx
        mov ah, al
        call TETR_TO_HEX
        xchg al,ah
        mov cl,4
        shr al,cl
        call TETR_TO_HEX
        pop cx
        ret
    BYTE_TO_HEX ENDP


    WORD_TO_HEX PROC
        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
    WORD_TO_HEX ENDP

CODE ENDS
     END MAIN_1
```

lab7.asm

```
DATA SEGMENT
    over1_name db "over_1.ovl", 0
    over2_name db "over_2.ovl", 0
    program DW 0
    data_mem db 43 DUP(0)
    mem_flag db 0
    cl_pos db 128 DUP(0)
    over_addres dd 0
    pspKeep DW 0

    EOF db 0dh, 0ah, '$'
    file_err_mes db 'File has not been found!', 0dh, 0ah, '$'
    path_err_mes db 'Path has not been found!', 0dh, 0ah, '$'
    mcb_err_mes db 'MCB has been crashed!', 0dh, 0ah, '$'
```

```
       no_mem_err_mes db 'Not enough memory!', 0dh, 0ah, '$'
       addres_err_mes db 'Invalid memory address', 0dh, 0ah, '$'

       func_load_err_mes db 'Invalid function!', 0dh, 0ah, '$'
       many_file_err_mes db 'Too many files have been opened!', 0dh, 0ah, '$'
       access_err_mes db 'Access error!', 0dh, 0ah, '$'
       mem_err_mes db 'Invalid memory!', 0dh, 0ah, '$'
       envir_err_mes db 'Invalid enviroment!', 0dh, 0ah, '$'

    mem_free_mes db 'Memory for calling module has been cleared!' , 0dh, 0ah, '$'
       alloc_mem__mes db  'Memory has been successfully allocated!', 0dh, 0ah, '$'
       suc_load_mes db  'Loaded has been successfully finished!', 0dh, 0ah, '$'

       data_end db 0
DATA ENDS

STACK SEGMENT STACK
       DW 128 DUP(?)
STACK ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:STACK

PRINT_MES PROC
       push ax

       mov ah, 09h
       int 21h

       pop ax
       ret
PRINT_MES ENDP

ALLOC_MEMORY PROC
       push ax
       push bx
       push cx
       push dx

       push dx
       mov dx, offset data_mem
       mov ah, 1ah
       int 21h

       pop dx
       mov cx, 0
       mov ah, 4eh
       int 21h

       jnc success_aloc

       cmp ax, 2
       je a_file_err

       cmp ax, 3
       je a_path_err

a_file_err:
       mov dx, offset file_err_mes
       call PRINT_MES
       jmp end_aloc

a_path_err:
       mov dx, offset path_err_mes
```

```
        call PRINT_MES
        jmp end_aloc

success_aloc:
        push di
        mov di, offset data_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 over_addres, ax
        mov dx, offset alloc_mem__mes
        call PRINT_MES

end_aloc:
        pop dx
        pop cx
        pop bx
        pop ax
        ret
ALLOC_MEMORY ENDP

FREE_MEM_FOR_OVER PROC
        push ax
        push bx
        push cx
        push dx

        mov ax, offset data_end
        mov bx, offset program_end

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

        jnc end_free
        mov mem_flag, 0

        cmp ax, 7
        je mcb_err

        cmp ax, 8
        je mem_err

        cmp ax, 9
        je addres_err

mcb_err:
        mov dx, offset mcb_err_mes
        call PRINT_MES
        jmp exit_free
```

```
mem_err:
	mov dx, offset no_mem_err_mes
	call PRINT_MES
	jmp exit_free


addres_err:
	mov dx, offset addres_err_mes
	call PRINT_MES
	jmp exit_free

end_free:
	mov mem_flag, 1
	mov dx, offset mem_free_mes
	call PRINT_MES

exit_free:
	pop dx
	pop cx
	pop bx
	pop ax
	ret
FREE_MEM_FOR_OVER ENDP

LOAD_OVER PROC
	push ax
	push bx
	push cx
	push dx
	push DS
	push es

	mov ax, DATA
	mov es, ax
	mov bx, offset over_addres
	mov dx, offset cl_pos
	mov ax, 4b03h
	int 21h

	jnc end_load

	cmp ax, 1
	je func_load_err

	cmp ax, 2
	je file_err

	cmp ax, 3
	je path_err

	cmp ax, 4
	je many_files_err

	cmp ax, 5
	je access_err

	cmp ax, 8
	je load_mem_err

	cmp ax, 10
	je envir_err

func_load_err:
	mov dx, offset EOF
```

```
        call PRINT_MES

        mov dx, offset func_load_err_mes
        call PRINT_MES
        jmp exit_load

file_err:
        mov dx, offset file_err_mes
        call PRINT_MES
        jmp exit_load

path_err:
        mov dx, offset EOF
        call PRINT_MES
        mov dx, offset path_err_mes
        call PRINT_MES
        jmp exit_load

many_files_err:
        mov dx, offset many_file_err_mes
        call PRINT_MES
        jmp exit_load

access_err:
        mov dx, offset access_err_mes
        call PRINT_MES
        jmp exit_load

load_mem_err:
        mov dx, offset mem_err_mes
        call PRINT_MES
        jmp exit_load

envir_err:
        mov dx, offset envir_err_mes
        call PRINT_MES
        jmp exit_load

end_load:
        mov dx, offset suc_load_mes
        call PRINT_MES

        mov ax, word ptr over_addres
        mov es, ax
        mov word ptr over_addres, 0
        mov word ptr over_addres+2, ax

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

exit_load:
        pop es
        pop DS
        pop dx
        pop cx
        pop bx
        pop ax
        ret
LOAD_OVER ENDP

PATH_MAKING PROC
        push ax
        push bx
```

```
        push cx
        push dx
        push di
        push si
        push es

        mov program, dx

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

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

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

        add bx, 2
        mov di, 0

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

end_path_loop:
        mov di, cx
        mov si, program

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


        pop es
        pop si
        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        ret
PATH_MAKING ENDP

LOAD_OVERLAY PROC
        push dx

        call PATH_MAKING
        mov dx, offset cl_pos
```

```
        call ALLOC_MEMORY
        call LOAD_OVER

        pop dx
        ret
LOAD_OVERLAY ENDP

MAIN_PROC PROC FAR
        push DS
        xor ax, ax
        push ax

        mov ax, DATA
        mov DS, ax
        mov pspKeep, es

        call FREE_MEM_FOR_OVER
        cmp mem_flag, 0
        je end_main

        mov dx, offset over1_name
        call LOAD_OVERLAY
        mov dx, offset EOF
        call PRINT_MES

        mov dx, offset over2_name
        call LOAD_OVERLAY

end_main:
        xor al, al
        mov ah, 4ch
        int 21h
MAIN_PROC ENDP

program_end:
CODE ENDS
END MAIN_PROC
```