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 _____ Янкин Д.О.

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.

Был написан программный модуль типа .EXE, который:

1) Освобождает память для загрузки оверлея.
2) Читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки.
3) Загружает и выполняет файл оверлейного сегмента.
4) Освобождает память, отведенную для оверлейного сегмента.
5) Повторяет действия для следующего оверлейного сегмента.

Запущена программа, когда оба модуля находятся в текущей директории. Смотреть рисунок 1.

Рисунок 1. Результат работы программы при модулях в одном каталоге

Запущена программа, когда оба модуля находятся в другой директории. Смотреть рисунок 2.

Рисунок 2. Результат работы программы при всех модулях в другом каталоге

Запущена программа, при отсутствующем первом оверлее. Программа завершается после первой ошибки. Смотреть рисунок 3.

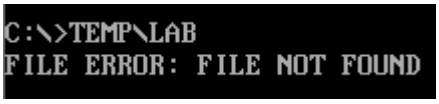

Рисунок 3. Результат работы программы при отсутствующем первом оверлее

Запущена программа, при отсутствующем втором оверлее. Смотреть рисунок 4.

Рисунок 4. Результат работы программы при отсутствующем втором оверлее

Контрольные вопросы.

1) Как должна быть устроен программа, если в качестве оверлейного сегмента использовать .COM модуль?

   Оверлейный сегмент необходимо вызывать со смещением 100h, иначе не будет сформирован PSP. Также необходимо сохранять значения регистров.

Выводы.

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

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ LAB.ASM

```
ASTACK     SEGMENT    STACK
           DB    100h  DUP(0)
ASTACK     ENDS

DATA SEGMENT
     MEM_REALLOC_ERROR_07_MESSAGE     DB    'MEM REALLOC ERROR: MCB
WAS DESTROYED$'
     MEM_REALLOC_ERROR_08_MESSAGE     DB    'MEM REALLOC ERROR: NOT
ENOUGH MEMORY$'
     MEM_REALLOC_ERROR_09_MESSAGE     DB    'MEM REALLOC ERROR:
WRONG MCB ADDRESS$'

     FILE_ERROR_02_MESSAGE            DB    'FILE ERROR: FILE NOT
FOUND$'
     FILE_ERROR_03_MESSAGE            DB    'FILE ERROR: PATH NOT
FOUND$'

     MEM_ALLOC_ERROR_MESSAGE               DB    'MEM ALLOC ERROR$'

     RUN_OVERLAY_ERROR_01_MESSAGE     DB    'LOAD ERROR: UNEXISTING
FUNCTION$'
     RUN_OVERLAY_ERROR_02_MESSAGE     DB    'LOAD ERROR: FILE NOT
FOUND$'
     RUN_OVERLAY_ERROR_03_MESSAGE     DB    'LOAD ERROR: PATH NOT
FOUND$'
     RUN_OVERLAY_ERROR_04_MESSAGE     DB    'LOAD ERROR: TOO MANY
FILES ARE OPEN$'
     RUN_OVERLAY_ERROR_05_MESSAGE     DB    'LOAD ERROR: NO ACCESS$'
     RUN_OVERLAY_ERROR_08_MESSAGE     DB    'LOAD ERROR: NOT ENOUGH
MEMORY$'

     PATH                                   DB    128 DUP(?)
```

```
	OVERLAY_01_NAME			DB	'OVERLAY1.OVL', 0
	OVERLAY_02_NAME			DB	'OVERLAY2.OVL', 0

	DTA					DB	43	DUP(?)
	OVERLAY_SEG				DW	0
	OVERLAY_ADDRESS			DD	0

	KEEP_SS					DW	0
	KEEP_SP					DW	0
DATA ENDS

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

	KEEP_DS					DW	0


	; Вывод строки по DS:DX. Логично
	PRINT_STRING PROC
		push ax

		mov ah, 09h
		int 21h

		pop ax
		ret
	PRINT_STRING ENDP


	; Перевод строки
	PRINT_ENDL PROC
		push ax
		push dx
```

```
        mov dl, 13
        mov ah, 02h
        int 21h
        mov dl, 10
        int 21h

        pop dx
        pop ax
        ret
    PRINT_ENDL ENDP


    SET_DTA PROC
        push  ax
        push  dx
        mov   dx, offset DTA
        mov   ah,1Ah
        int   21h
        pop         dx
        pop         ax
    SET_DTA ENDP


    FREE_EXTRA_MEMORY PROC
        push  ax
        push  bx
        push  cx
        push  dx

        ; Расстояние от начала PSP до начала CODE в параграфах в
AX
        mov         ax, cs
        mov         bx,   es
        sub         ax, bx
```

```
            ; Расстояние от начала CODE до его конца в байтах --> в
параграфах
            mov         bx, offset PROGRAM_END
            mov         cl, 4
            shr         bx, cl
            add         bx, 1

            ; Суммирование --> в BX
            add         bx, ax

            ; Уменьшение обьема памяти до того, что в BX
            mov   ah, 4Ah
            int   21h
            jnc         FREE_EXTRA_MEMORY_RET


            cmp   ax, 7
            je          MEM_REALLOC_ERROR_07
            cmp   ax, 8
            je          MEM_REALLOC_ERROR_08
            cmp   ax, 9
            je          MEM_REALLOC_ERROR_09

            MEM_REALLOC_ERROR_07:
            mov   dx, offset MEM_REALLOC_ERROR_07_MESSAGE
            jmp         MEM_REALLOC_ERROR_EXIT

            MEM_REALLOC_ERROR_08:
            mov   dx, offset MEM_REALLOC_ERROR_08_MESSAGE
            jmp         MEM_REALLOC_ERROR_EXIT

            MEM_REALLOC_ERROR_09:
            mov   dx, offset MEM_REALLOC_ERROR_09_MESSAGE
```

```
	MEM_REALLOC_ERROR_EXIT:
	call  PRINT_STRING
	call  PRINT_ENDL
	mov		ax, 4C00h
	int		21h


	FREE_EXTRA_MEMORY_RET:
	pop		dx
	pop		cx
	pop   bx
	pop   ax
	ret
FREE_EXTRA_MEMORY ENDP


SET_PATH PROC
	push  ax
	push  si
	push  di
	push  es

	; Взятие пути программы
	mov   es, es:[2Ch]
	mov   si, 0

	ENVIROMENT_STRING_LOOP:
	mov   ah, es:[si]
	cmp   ah, 00h
	je		ENVIROMENT_STRING_ENDL
	inc   si
	jmp   ENVIROMENT_STRING_LOOP
```

```
ENVIROMENT_STRING_ENDL:
inc   si
mov   ah, es:[si]
cmp   ah, 00h
jne   ENVIROMENT_STRING_LOOP
add   si, 03h

mov   di, offset PATH

PATH_LOOP:
mov   ah, es:[si]
mov   [di], ah
cmp   ah, 00h
je          PATH_ENDL
inc   si
inc   di
jmp   PATH_LOOP

PATH_ENDL:
sub         di, 7
mov         si, dx

NAME_LOOP:
mov         ah, [si]
mov         [di], ah
cmp         ah, 00h
je          SET_PATH_RET
inc         si
inc         di
jmp         NAME_LOOP

SET_PATH_RET:
pop         es
pop         di
```

```
        pop         si
        pop         ax
        ret
SET_PATH ENDP


ALLOCATE_MEMORY PROC
        push  ax
        push  bx
        push  cx
        push  dx

        mov         dx, offset PATH
        mov   cx, 0
        mov   ax, 4E00h
        int   21h

        jnc         ALLOCATE_MEMORY_FILE_SUCCESS


        cmp         ax, 2
        je          ALLOCATE_MEMORY_FILE_ERROR_02
        cmp         ax, 3
        je          ALLOCATE_MEMORY_FILE_ERROR_03

        ALLOCATE_MEMORY_FILE_ERROR_02:
        mov         dx, offset FILE_ERROR_02_MESSAGE
        jmp         ALLOCATE_MEMORY_FILE_ERROR_MESSAGE

        ALLOCATE_MEMORY_FILE_ERROR_03:
        mov         dx, offset FILE_ERROR_03_MESSAGE
        jmp         ALLOCATE_MEMORY_FILE_ERROR_MESSAGE

        ALLOCATE_MEMORY_FILE_ERROR_MESSAGE:
```

```
call  PRINT_STRING
call  PRINT_ENDL
mov         ax, 4C00h
int         21h


ALLOCATE_MEMORY_FILE_SUCCESS:
mov   bx, offset DTA
mov   ax, [bx+1Ch]    ; Старшее
mov   bx, [bx+1Ah]    ; Младшее

mov   cl, 12
shl         ax, cl
mov         cl, 4
shr         bx, cl

add   bx, ax
inc   bx

mov   ah, 48h
int   21h
jnc         ALLOCATE_MEMORY_RET


mov         dx, offset MEM_ALLOC_ERROR_MESSAGE
call  PRINT_STRING
call  PRINT_ENDL

mov         ax, 4C00h
int         21h


ALLOCATE_MEMORY_RET:
mov   OVERLAY_SEG, ax
```

```
        pop         dx
        pop         cx
        pop         bx
        pop         ax
        ret
ALLOCATE_MEMORY ENDP


RUN_OVERLAY PROC
        push  ax
        push  bx
        push  dx
        push  es

        push  es
        mov         KEEP_DS, ds
        mov         KEEP_SS, ss
        mov         KEEP_SP, sp

        mov         bx, seg OVERLAY_SEG
        mov         es, bx
        mov         bx, offset OVERLAY_SEG

        mov         dx, offset PATH
        mov   ax, 4B03h
        int   21h

        mov         ds, KEEP_DS
        mov         ss, KEEP_SS
        mov         sp, KEEP_SP
        pop         es

        jnc         RUN_OVERLAY_OK
```

```
cmp         ax, 1
je          RUN_OVERLAY_ERROR_01
cmp         ax, 2
je          RUN_OVERLAY_ERROR_02
cmp         ax, 3
je          RUN_OVERLAY_ERROR_03
cmp         ax, 4
je          RUN_OVERLAY_ERROR_04
cmp         ax, 5
je          RUN_OVERLAY_ERROR_05
cmp         ax, 8
je          RUN_OVERLAY_ERROR_08

RUN_OVERLAY_ERROR_01:
mov         dx, offset RUN_OVERLAY_ERROR_01_MESSAGE
jmp         RUN_OVERLAY_ERROR_PRINT

RUN_OVERLAY_ERROR_02:
mov         dx, offset RUN_OVERLAY_ERROR_02_MESSAGE
jmp         RUN_OVERLAY_ERROR_PRINT

RUN_OVERLAY_ERROR_03:
mov         dx, offset RUN_OVERLAY_ERROR_03_MESSAGE
jmp         RUN_OVERLAY_ERROR_PRINT

RUN_OVERLAY_ERROR_04:
mov         dx, offset RUN_OVERLAY_ERROR_04_MESSAGE
jmp         RUN_OVERLAY_ERROR_PRINT

RUN_OVERLAY_ERROR_05:
mov         dx, offset RUN_OVERLAY_ERROR_05_MESSAGE
jmp         RUN_OVERLAY_ERROR_PRINT
```

```
	RUN_OVERLAY_ERROR_08:
	mov	dx, offset RUN_OVERLAY_ERROR_08_MESSAGE

	RUN_OVERLAY_ERROR_PRINT:
	call	PRINT_STRING
	call	PRINT_ENDL
	mov	ax, 4C00h
	int	21h


	RUN_OVERLAY_OK:
	mov	ax, OVERLAY_SEG
	mov	word ptr OVERLAY_ADDRESS+2, ax
	call	OVERLAY_ADDRESS

	mov	ax, OVERLAY_SEG
	mov	es, ax
	mov	ah, 49h
	int	21h

	pop	es
	pop	dx
	pop	bx
	pop	ax
	ret
RUN_OVERLAY ENDP


MAIN PROC
	mov	ax, DATA
	mov	ds, ax

	call	SET_DTA
```

```
            call  FREE_EXTRA_MEMORY

            mov   dx, offset OVERLAY_01_NAME
            call  SET_PATH
            call  ALLOCATE_MEMORY
            call  RUN_OVERLAY

            sub   bx,bx

            mov   dx, offset OVERLAY_02_NAME
            call  SET_PATH
            call  ALLOCATE_MEMORY
            call  RUN_OVERLAY

            mov   ax,4C00h
            int   21h
      MAIN ENDP

      PROGRAM_END:
CODE  ENDS
END MAIN
```