МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по практической работе № 7

по дисциплине «Операционные системы»

Тема: Построение модуля оверлейной структуры

Студентка гр. 9383 _____ Лапина А.А.

Преподаватель _____ Ефремов М. А.

Санкт-Петербург

2021

Цель работы.

Исследовать возможность построения загрузочного модуля оверлейной структуры, структуру оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов.

Постановка задачи.

Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет следующие функции:

1. освобождает память для загрузки оверлеев;
2. читает размер файла оверлея и запрашивает объём памяти, достаточный для его загрузки;
3. загружает и выполняет оверлейный сегмент;
4. освобождает память, отведённую для оверлейного сегмента;
5. выполняет аналогичные действия для следующего оверлейного сегмента.

Также необходимо написать и отладить оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

Запустить отлаженное приложение. Оверлейные сегменты должны загружаться с одного и того же адреса, перекрывая друг друга.

Запустить приложение из другого каталога. Приложение должно быть выполнено успешно.

Запустить приложение в случае, когда одного оверлея нет в каталоге. Приложение должно закончиться аварийно.

Выполнение работы.

Для выполнения данной работы были реализованы следующие функции:

* println для вывода строки на экран;
* num2dec для записи в строку числа, лежащего в регистре AL;
* free для очистки неиспользуемой модулем памяти;
* setp для создания блока параметров;
* getpath для получения пути вызываемого модуля;

* callovl для вызова оверлейного модуля;

* main для выполнения поставленной в данной лабораторной работе задачи.

Для вывода информации на экран были созданы следующие строки:

* unknown, хранящая в себе строку 'Unknown error$';

* ferr7, хранящая в себе строку 'Error: memory control block destroyed$';

* ferr8, хранящая в себе строку 'Error: not enough memory to execute the function$';

* ferr9, хранящая в себе строку 'Error: invalid memory block address$';

* serr2, хранящая в себе строку 'Error: file could not be find$';

* serr3, хранящая в себе строку 'Error: path could not be find$';

* cerr1, хранящая в себе строку 'Error: function is not exists$';

* cerr2, хранящая в себе строку 'Error: file could not be found$';

* cerr3, хранящая в себе строку 'Error: path could not be found$';

* cerr4, хранящая в себе строку 'Error: too many open files$';

* cerr5, хранящая в себе строку 'Error: no access$';

* cerr8, хранящая в себе строку 'Error: no memory$';

* cerrA, хранящая в себе строку 'Error: wrong environment string$'.

Выполнение программы начинается с вызова функции free для освобождения неиспользуемой программой памяти. Для этого с помощью функции 4Ah прерывания 21h выделяется необходимое программе количество памяти. В случае, если при этом возникла какая-либо ошибка, программа выводит соответствующее сообщение об ошибке и завершается.

После этого вызывается функция setp, которая заполняет выделенный в сегменте данных блок параметров.

С помощью функции getpath считывается в строку путь до текущего файла, после чего имя файла меняется на имя вызываемых оверлейных модулей.

Функция callovl заполняет выделенный в памяти буфер DTA, выделяет память под оверлейный модуль, сохраняет сегментные регистры DS и ES, а

также регистры, отвечающие за стек. Устанавливаются регистры ES:BX так, чтобы они указывали на блок параметров, после чего вызывается программный модуль.

После выполнения модуля регистры возвращаются, проверяется работа модуля.

Разработанный программный код см. в приложении А.

Результаты работы программы представлены на рисунках 1-4.

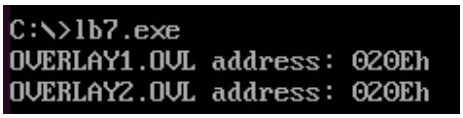

Рисунок 1 – Запуск приложения

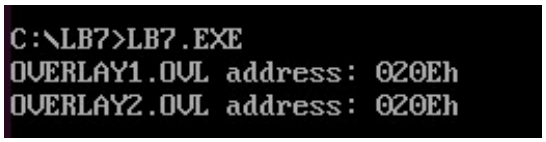

Рисунок 2 – Запуск приложения из другого каталога

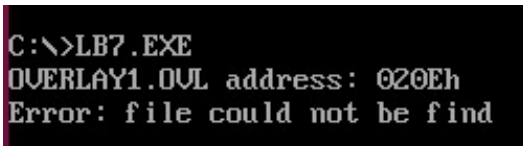

Рисунок 3 – Запуск приложения при отсутствии overlay2.oul

Выводы.

В ходе работы были исследованы возможность построения загрузочного модуля оверлейной структуры, структура оверлейного сегмента и способ dosbox загрузки и выполнения оверлейных сегментов.

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

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

При обращении к оверлейному .COM модулю необходимо обратиться к сегменту, смещённому на 100h относительно начала модуля, ввиду наличия PSP.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb7.asm

```
CODE    SEGMENT
ASSUME  CS:CODE, DS:DATA

PRINTLN PROC
        PUSH    AX
        PUSH    DX
        MOV     AH, 09H
        INT     21H
        MOV     AH, 02H
        MOV     DL, 0AH
        INT     21H
        MOV     DL, 0DH
        INT     21H
        POP     DX
        POP     AX
        RET
PRINTLN ENDP

NUM2DEC PROC
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        MOV     BX, 10
        XOR     CX, CX
        XOR     DX, DX
DIV10:  DIV     BL
        MOV     DL, AH
        PUSH    DX
        XOR     AH, AH
        INC     CX
        XOR     DX, DX
        CMP     AL, 0
        JNE     DIV10
LOOP10: POP     DX
        XOR     DH, DH
        ADD     DL, 30H
        MOV     BYTE PTR [SI], DL
```

```
        INC     SI
        LOOP    LOOP10
        POP     SI
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        RET
NUM2DEC ENDP

FREE    PROC
        PUSH    AX
        PUSH    BX
        PUSH    DX
        XOR     DX, DX
        MOV     AX, OFFSET ENDPROG
        ADD     AX, OFFSET ENDDATA
        ADD     AX, 300H
        MOV     BX, 16
        DIV     BX
        MOV     BX, AX
        INC     BX
        MOV     AH, 4AH
        INT     21H
        JNC     ENDF
        CMP     AX, 7
        JE      EFREE7
        CMP     AX, 8
        JE      EFREE8
        CMP     AX, 9
        JE      EFREE9
        JMP     EFREEU
EFREE7: MOV     DX, OFFSET FERR7
        JMP     EPRINT
EFREE8: MOV     DX, OFFSET FERR8
        JMP     EPRINT
EFREE9: MOV     DX, OFFSET FERR9
        JMP     EPRINT
EFREEU: MOV     DX, OFFSET UNKNOWN
EPRINT: CALL    PRINTLN
        MOV     FLAG, 1
ENDF:   POP     DX
        POP     BX
        POP     AX
```

```
        RET
FREE    ENDP

SETP    PROC
        PUSH    AX
        MOV     AX, ES:[2CH]
        MOV     PARAM, AX
        MOV     PARAM + 2, ES
        MOV     PARAM + 4, 80H
        POP     AX
        RET
SETP    ENDP

GETPATH PROC
        PUSH    DX
        PUSH    DI
        PUSH    SI
        PUSH    ES
        XOR     DI, DI
        MOV     ES, ES:[2CH]
        MOV     DL, ES:[DI]
        JMP     CHECK
NEXTC:  INC     DI
        MOV     DL, ES:[DI]
CHECK:  CMP     DL, 00H
        JNE     NEXTC
        INC     DI
        MOV     DL, ES:[DI]
        CMP     DL, 00H
        JNE     NEXTC
        XOR     SI, SI
        ADD     DI, 3
GETC:   MOV     DL, ES:[DI]
        CMP     DL, 00H
        JE      GETF
        MOV     BYTE PTR OPATH1[SI], DL
        MOV     BYTE PTR OPATH2[SI], DL
        INC     DI
        INC     SI
        JMP     GETC
GETF:   DEC     SI
        MOV     DL, OPATH1[SI]
        CMP     DL, '\'
        JNE     GETF
```

```
        INC     SI
        XOR     DI, DI
ADDP:   MOV     DL, ONAME1[DI]
        MOV     DH, ONAME2[DI]
        CMP     DL, '$'
        JE      ENDG
        MOV     OPATH1[SI], DL
        MOV     OPATH2[SI], DH
        INC     DI
        INC     SI
        JMP     ADDP
ENDG:   MOV     OPATH1[SI], 00H
        MOV     OPATH2[SI], 00H
        POP     ES
        POP     SI
        POP     DI
        POP     DX
        RET
GETPATH ENDP

CALLOVL PROC
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        MOV     AX, 1A00H
        MOV     DX, OFFSET DTA
        INT     21H
        MOV     AX, 4E00H
        MOV     CX, 0
        MOV     DX, FILE
        INT     21H
        JNC     OVL
        CMP     AX, 3
        JE      ESIZE3
        CMP     AX, 2
        JE      ESIZE2
ESIZE2: MOV     DX, OFFSET SERR2
        JMP     ERREND
ESIZE3: MOV     DX, OFFSET SERR3
        JMP     ERREND
OVL:    MOV     SI, OFFSET DTA
        MOV     BX, [SI + 1AH]
```

```
        MOV     CL, 4
        SHR     BX, CL
        MOV     AX, [SI + 1CH]
        MOV     CL, 12
        SHL     AX, CL
        ADD     BX, AX
        ADD     BX, 1
        MOV     DX, FILE
        MOV     AX, 4800H
        INT     21H
        MOV     OBLOCK, AX
        PUSH    DS
        PUSH    ES
        MOV     KEEPSS, SS
        MOV     KEEPSP, SP
        MOV     AX, DS
        MOV     ES, AX
        MOV     BX, OFFSET OBLOCK
        MOV     DX, FILE
        MOV     AX, 4B03H
        INT     21H
        MOV     DX, KEEPSP
        MOV     SP, DX
        MOV     SS, KEEPSS
        POP     ES
        POP     DS
        JNC     OK
        CMP     AX, 1
        JE      ECALL1
        CMP     AX, 2
        JE      ECALL2
        CMP     AX, 3
        JE      ECALL3
        CMP     AX, 4
        JE      ECALL4
        CMP     AX, 5
        JE      ECALL5
        CMP     AX, 8
        JE      ECALL8
        CMP     AX, 10
        JE      ECALLA
        JMP     ECALLU
ECALL1: MOV     DX, OFFSET CERR1
        JMP     ERREND
```

```
ECALL2: MOV     DX, OFFSET CERR2
        JMP     ERREND
ECALL3: MOV     DX, OFFSET CERR3
        JMP     ERREND
ECALL4: MOV     DX, OFFSET CERR4
        JMP     ERREND
ECALL5: MOV     DX, OFFSET CERR5
        JMP     ERREND
ECALL8: MOV     DX, OFFSET CERR8
        JMP     ERREND
ECALLA: MOV     DX, OFFSET CERRA
        JMP     ERREND
ECALLU: MOV     DX, OFFSET UNKNOWN
        JMP     ERREND
OK:     PUSH    ES
        MOV     AX, OBLOCK
        MOV     ES, AX
        MOV     WORD PTR OVLBEG + 2, AX
        CALL    DS:[OVLBEG]
        MOV     ES, AX
        MOV     AX, 4900H
        INT     21H
        POP     ES
        JMP     OEND
ERREND: CALL    PRINTLN
        MOV     FLAG, 1
OEND:   POP     SI
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        RET
CALLOVL ENDP

MAIN    PROC FAR
        MOV     AX, DATA
        MOV     DS, AX
        CALL    FREE
        CMP     FLAG, 0
        JNE     FIN
        CALL    SETP
        CALL    GETPATH
        MOV     DX, OFFSET OPATH1
        MOV     FILE, DX
```

```
        CALL    CALLOVL
        CMP     FLAG, 0
        JNE     FIN
        MOV     DX, OFFSET OPATH2
        MOV     FILE, DX
        CALL    CALLOVL
; --- END ---
FIN:    MOV     AX, 4C00H
        INT     21H
MAIN    ENDP

ENDPROG:
CODE    ENDS

DATA    SEGMENT
PARAM   DW 7 DUP (0)
DTA     DB 43 DUP (0)
ONAME1  DB 'OVERLAY1.OVL$'
ONAME2  DB 'OVERLAY2.OVL$'
OPATH1  DB 64 DUP (0), '$'
OPATH2  DB 64 DUP (0), '$'
FILE    DW 0
OBLOCK  DW 0
FLAG    DB 0
KEEPSS  DW 0
KEEPSP  DW 0
UNKNOWN DB 'UNKNOWN ERROR$'
FERR7   DB 'ERROR: MEMORY CONTROL BLOCK DESTROYED$'
FERR8   DB 'ERROR: NOT ENOUGH MEMORY TO EXECUTE THE FUNCTION$'
FERR9   DB 'ERROR: INVALID MEMORY BLOCK ADDRESS$'
SERR2   DB 'ERROR: FILE COULD NOT BE FIND$'
SERR3   DB 'ERROR: PATH COULD NOT BE FIND$'
CERR1   DB 'ERROR: FUNCTION IS NOT EXISTS$'
CERR2   DB 'ERROR: FILE COULD NOT BE FOUND$'
CERR3   DB 'ERROR: PATH COULD NOT BE FOUND$'
CERR4   DB 'ERROR: TOO MANY OPEN FILES$'
CERR5   DB 'ERROR: NO ACCESS$'
CERR8   DB 'ERROR: NO MEMORY$'
CERRA   DB 'ERROR: WRONG ENVIRONMENT STRING$'
OVLBEG  DD 0
ENDDATA DB ?
DATA    ENDS

STACK   SEGMENT STACK
```

```
        DB 256 DUP (?)
STACK   ENDS

        END     MAIN
```

Название файла: overlay1.asm

```
code    segment
assume  CS:code

main    proc far
        push    AX
        push    DX
        push    DS
        push    SI
        mov     DX, CS
        mov     DS, DX
        mov     SI, offset str
        add     SI, 22
        call    reg2hex
        mov     DX, offset str
        mov     AH, 09h
        int     21h
        pop     SI
        pop     DS
        pop     DX
        pop     AX
        retf
main    endp

reg2hex proc
        push    AX
        push    BX
        push    CX
        push    DX
        mov     BX, 0F000h
        mov     DL, 12
        mov     CX, 4
nloop:  push    CX
        push    AX
        and     AX, BX
        mov     CL, DL
        shr     AX, CL
        cmp     AL, 9
```

```
        ja      lttr
        add     AL, 30h
        jmp     ok
lttr:   add     AL, 37h
ok:     mov     byte ptr [SI], AL
        inc     SI
        pop     AX
        mov     CL, 4
        shr     BX, CL
        sub     DL, 4
        pop     CX
        loop    nloop
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        ret
reg2hex endp

str     db 'OVERLAY1.OVL address: 0000h', 0Ah, 0Dh, '$'

code    ends

        end     main
```

Название файла: overlay2.asm

```
code    segment
assume  CS:code

main    proc far
        push    AX
        push    DX
        push    DS
        push    SI
        mov     DX, CS
        mov     DS, DX
        mov     SI, offset str
        add     SI, 22
        call    reg2hex
        mov     DX, offset str
        mov     AH, 09h
```

```
        int     21h
        pop     SI
        pop     DS
        pop     DX
        pop     AX
        retf
main    endp

reg2hex proc
        push    AX
        push    BX
        push    CX
        push    DX
        mov     BX, 0F000h
        mov     DL, 12
        mov     CX, 4
nloop:  push    CX
        push    AX
        and     AX, BX
        mov     CL, DL
        shr     AX, CL
        cmp     AL, 9
        ja      lttr
        add     AL, 30h
        jmp     ok
lttr:   add     AL, 37h
ok:     mov     byte ptr [SI], AL
        inc     SI
        pop     AX
        mov     CL, 4
        shr     BX, CL
        sub     DL, 4
        pop     CX
        loop    nloop
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        ret
reg2hex endp

str     db 'OVERLAY2.OVL address: 0000h', 0Ah, 0Dh, '$'
```

```
code    ends

        end     main
```