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

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Операционные системы»

ТЕМА: ПОСТРОЕНИЕ МОДУЛЯ ДИНАМИЧЕСКОЙ СТРУКТУРЫ.

Студентка гр. 0382 _____ Чегодаева Е.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры. В отличии от предыдущих лабораторных работ в этой работе рассматривается приложение, состоящее из нескольких модулей, а не из одного модуля простой структуры. В этом случае разумно предположить, что все модули приложения находятся в одном каталоге и полный путь в этот каталог можно взять из среды, как это делалось в работе 2. Понятно, что такое приложение должно запускаться в соответствии со стандартами ОС.

В работе исследуется интерфейс между вызывающим и вызываемым модулями по управлению и по данным. Для запуска вызываемого модуля используется функция 4B00h прерывания int 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Задание.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции:

1) Подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится он сам. Вызываемому модулю передается новая среда, созданная вызывающим модулем и новая командная строка.

2) Вызываемый модуль запускается с использованием загрузчика.

3) После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы. Необходимо проверять причину завершения и, в зависимости от значения, выводить соответствующее сообщение. Если причина завершения 0, то выводится код завершения.

В качестве вызываемой программы необходимо взять программу ЛР 2, которая распечатывает среду и командную строку. Эту программу следует немного модифицировать, вставив перед выходом из нее обращение к функции ввода символа с клавиатуры. Введенное значение записывается в регистр AL и затем происходит обращение к функции выхода 4Ch прерывания int 21h.

Шаг 2. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Введите произвольный символ из числа A-Z. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

Шаг 3. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Введите комбинацию символов Ctrl-C. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

Шаг 4. Запустите отлаженную программу, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули. Повторите ввод комбинаций клавиш. Занесите полученные данные в отчет.

Шаг 5. Запустите отлаженную программу, когда модули находятся в разных каталогах. Занесите полученные данные в отчет.

Выполнение работы.

Шаг 1.

Был написан и отлажен программный модуль типа .EXE, который подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится сам и запускает его, с использованием загрузчика. Программа проверяет корректность работы загрузчика и выполнения вызываемой программы.

В качестве вызываемого модуля была взята программа ЛР№2 (модифицированная, согласно требованиям), которая распечатывает среду и командную строку.

Функции, реализованные в работе:

- CLEAN_MEMORY — Освобождение неиспользуемой памяти;

- GET_P — Получение пути к вызываемому модулю;
- PARAMETERS — Создание блока параметров;
- DEAL — Запуск вызываемого модуля;
- PRINT — Осуществление вывода;
- BYTE_TO_HEX, TETR_TO_HEX — Вспомогательные функции для перевода в 16-тиричную систему счисления.

Шаг 2. Программа lb6.exe была запущена из каталога с разработанными модулями и введён символ A:

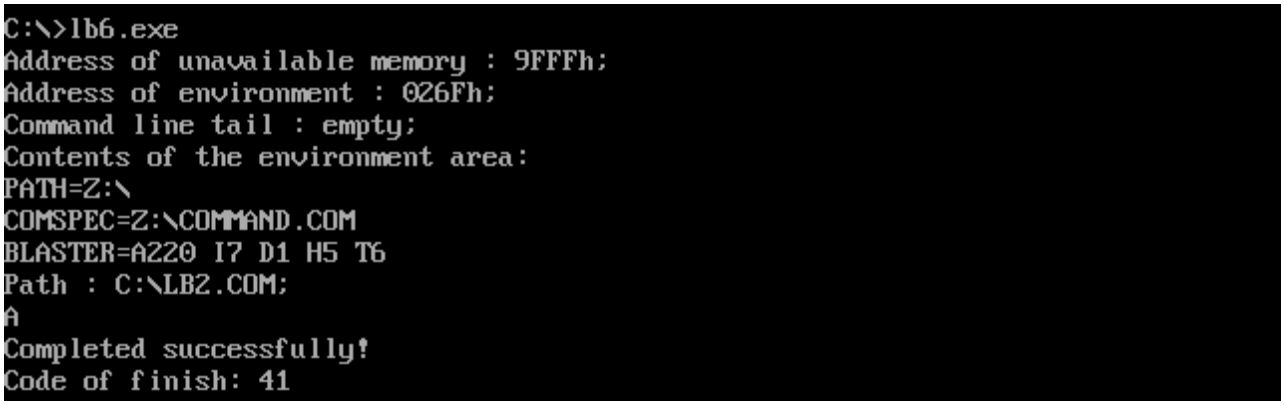

Рисунок 1 — Результат загрузки lb6.exe из текущего каталога + “A”

Шаг 3. Следом программа lb6.exe была вновь запущена из того же каталога, введена комбинация символов Ctrl+C:

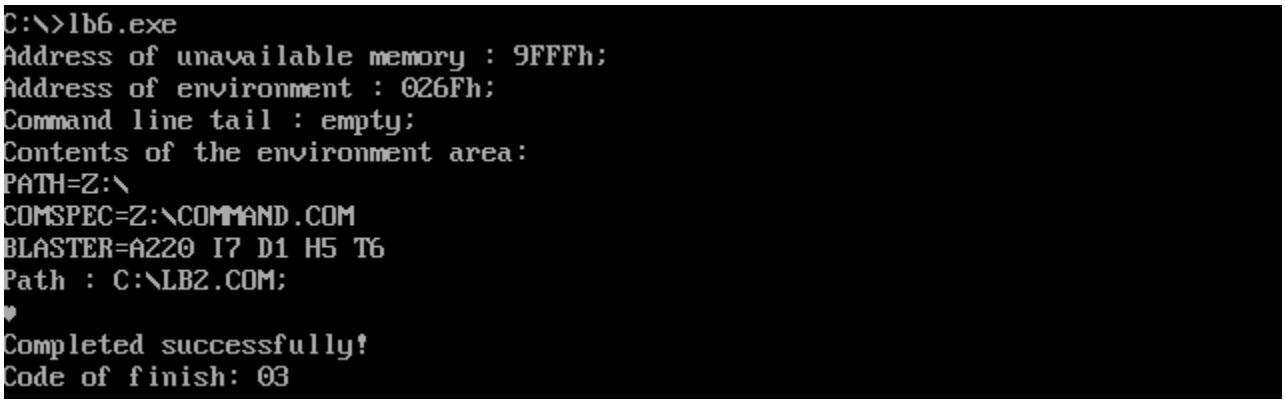

Рисунок 2 — Результат загрузки lb6.exe из текущего каталога + “ Ctrl+C”

Исходя из вывода программы видно, что результат аналогичен шагу 2, связано это с тем, что DosBox распознаёт комбинацию Ctrl+C как обычный символ (сердечко).

Шаг 4. Далее загрузочные модули были перемещены в директорию . /folder/ и lb6.exe была запущена оттуда, при этом был введён символ F и затем комбинация Ctrl+C :

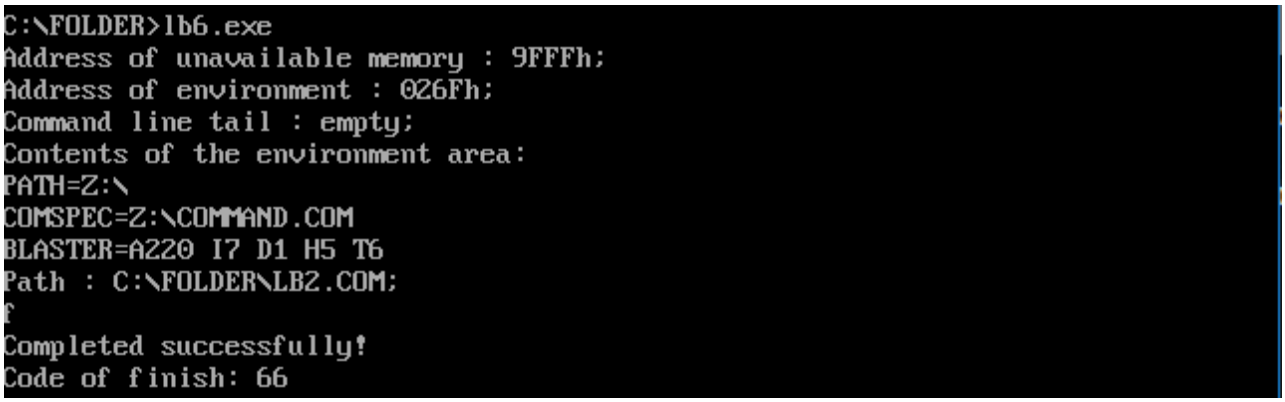

Рисунок 3 — Результат загрузки lb6.exe из каталога /folder/ + “ f”

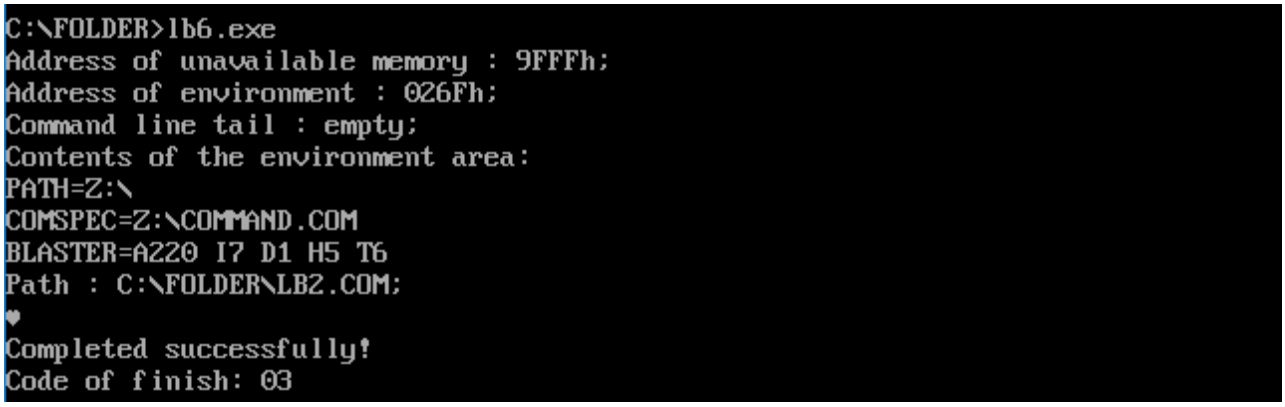

Рисунок 4 — Результат загрузки lb6.exe из каталога /folder/ + “ Ctrl+C ”

Шаг 5. Затем модуль lb2.com был перемещён обратно в корневую директорию, а программа lb6.exe осталась в каталоге /folder/ и была запущена оттуда:

Рисунок 5 — Результат загрузки lb6.exe при расположении модулей в разных директориях

Исходный код программ см. в приложении А

Контрольные вопросы.

1) Как реализовано прерывание Ctrl-C?

- Ответ: При прерывании Ctrl-C происходит обращение к прерыванию int 23h. Стандартный обработчик прерывания 23h завершает выполнение программы.

2) В какой точке заканчивается вызываемая программа, если код причины завершения 0?

- Ответ: В случае, если код причины завершения равен 0, то программа заканчивается при достижении вызова функции 4Ch прерывания 21h.

3) В какой точке заканчивается вызываемая программа по прерыванию Ctrl-C?

- Ответ: Вызываемая программа по прерыванию Ctrl-C заканчивается в месте вызова функции 01h прерывания 21h, то есть в месте, где ожидается ввод символа.

Выводы.

Была исследована возможность построения загрузочного модуля динамической структуры.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lb6.asm

```
AStack SEGMENT STACK
   DW 100 DUP(?)
AStack ENDS

DATA SEGMENT
ERRMEM db 'Memory cleanup error: $'
ERR_MCB db 'MCB is destroyed', 0DH, 0AH, '$'
NO_MEM db 'Deficiency memory', 0DH,0AH,'$'
ERR_ADR db 'error of the address', 0DH,0AH,'$'
ERR_FUN db 'Feature number error!', 0DH, 0AH, '$'
ERR_FILE db 'Error of the file!', 0DH, 0AH, '$'
ERR_DISK db 'Error of the disk!', 0DH, 0AH, '$'
ERR_ENV db 'Error of env!', 0DH, 0AH, '$'
ERR_FORM db 'Error of format!', 0DH, 0AH, '$'
ERR_DEVICE db 'Device error!', 0DH, 0AH, '$'
END_CTRL db 'End ctrl', 0DH, 0AH, '$'
ERR_RES db 'End 31h', 0DH, 0AH, '$'
CODE_ELEM db 'Code of finish: $'
SUCCESS db 'Completed successfully!', 0DH,0AH,'$'
END_S db  0DH, 0AH, '$'
PARAM dw 0
  dd 0
  dd 0
  dd 0
PATH db 50h dup ('$')
KEEP_SS dw 0
KEEP_SP dw 0
DATA ENDS

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

PRINT PROC NEAR
        push AX
        mov AH, 09h
        int 21h
        pop AX
        ret
PRINT ENDP

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
        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

CLEAN_MEMORY PROC
        mov AX, AStack
        mov BX, ES
        sub AX, BX
        add AX, 10h
        mov BX, AX
        mov AH, 4Ah
        int 21h
        jnc FINAL

        mov DX, offset ERRMEM
        call PRINT
        cmp AX, 7
        mov DX, offset ERR_MCB
        je PRINT_MEM
        cmp AX, 8
        mov DX, offset NO_MEM
        je PRINT_MEM
        cmp AX, 9
        mov DX, offset ERR_ADR

PRINT_MEM:
        call PRINT
        xor AL, AL
        mov AH, 4Ch
        int 21H
FINAL:
        ret
CLEAN_MEMORY ENDP
```

```
GET_P PROC
        mov AX, AStack
        sub AX, CODE
        add AX, 100h
        mov BX, AX
        mov AH, 4ah
        int 21h
        jnc step_1
        call DEAL
step_1:
        call PARAMETERS
        mov ES, ES:[2ch]
        mov BX,-1
step_2:
        add BX, 1
        cmp word ptr ES:[BX], 0000h
        jne step_2
        add BX, 4
        mov SI,-1
step_3:
        add SI, 1
        mov AL, ES:[BX+SI]
        mov PATH[SI], AL
        cmp byte ptr ES:[BX+SI], 00h
        jne step_3
        add SI, 1
step_4:
        mov PATH[SI], 0
        sub SI, 1
        cmp byte ptr ES:[BX+SI],'\'
        jne step_4
        add SI, 1
        mov PATH[SI],'l'
        add SI, 1
        mov PATH[SI],'b'
        add SI, 1
        mov PATH[SI],'2'
        add SI, 1
        mov PATH[SI],'.'
        add SI, 1
        mov PATH[SI],'C'
        add SI, 1
        mov PATH[SI],'O'
        add SI, 1
        mov PATH[SI],'M'
        ret
GET_P ENDP

PARAMETERS PROC
```

```
        mov AX, ES:[2Ch]
        mov PARAM, AX
        mov PARAM+2, ES
        mov PARAM+4, 80h
        ret
PARAMETERS ENDP

DEAL PROC
        mov DX, offset PATH
        xor CH, CH
        mov CL, ES:[80h]
        cmp CX, 0
        je UNTAIL
        mov SI, CX
        push SI
lp:
        mov AL, ES:[81h+SI]
        mov [offset PATH+SI-1], AL
        sub SI, 1
        loop lp
        pop SI
        mov [PATH+SI-1], 0
        mov DX,offset PATH
UNTAIL:
        push DS
        pop ES
        mov BX, offset PARAM
        mov KEEP_SP, SP
        mov KEEP_SS, SS
        mov AX, 4b00h
        int 21h
        jnc FIN
        push AX
        mov AX, DATA
        mov DS, AX
        pop AX
        mov SS, KEEP_SS
        mov SP, KEEP_SP

        cmp AX,1
        mov DX, offset ERR_FUN
        je PRINT_DEAL
        cmp ax,2
        mov DX, offset ERR_FILE
        je PRINT_DEAL
        cmp ax,5
        mov DX, offset ERR_DISK
        je PRINT_DEAL
        cmp ax,8
```

```
        mov DX, offset NO_MEM
        je PRINT_DEAL
        cmp ax,10
        mov DX, offset ERR_ENV
        je PRINT_DEAL
        cmp ax,11
        mov DX, offset ERR_FORM
PRINT_DEAL:
        call PRINT
        xor AL, AL
        mov AH, 4Ch
        int 21H
FIN:
        mov DX, offset END_S
        call PRINT
        mov AX, 4d00h
        int 21h
        cmp AH, 0
        mov DX, offset SUCCESS
        je REASONS
        cmp ah,1
        mov DX, offset END_CTRL
        je REASONS
        cmp ah,2
        mov DX, offset ERR_DEVICE
        je REASONS
        cmp ah,3
        mov DX, offset ERR_RES
REASONS:
        call PRINT
        mov DX, offset CODE_ELEM
        call PRINT
        call BYTE_TO_HEX
        push AX
        mov AH, 02h
        mov DL, AL
        int 21h
        pop AX
        xchg AH, AL
        mov AH, 02h
        mov DL, AL
        int 21h
        mov DX, offset END_S
        call PRINT
        ret
DEAL ENDP

Main PROC FAR
        mov AX, DATA
```

```
        mov DS, AX
        call CLEAN_MEMORY
        call GET_P
        call DEAL
        xor AL, AL
        mov AH, 4Ch
        int 21h
Main ENDP

CODE ENDS
   END Main
```

Название файла: lb2.asm

```
TESTPC SEGMENT
        ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
        ORG    100H
START: jmp BEGIN
; Д А Н Н Ы Е
UNV db 'Address of unavailable memory :    h;', 0DH, 0AH, '$'
ENV db 'Address of environment :    h;', 0DH, 0AH, '$'
CMD db 'Command line tail :', '$'
CMD_Emp db ' empty;', 0DH, 0AH, '$'
END_C db ';', 0DH, 0AH, '$'
ENV_Cnt db 'Contents of the environment area:',0DH, 0AH,'$'
PATH db 'Path :', '$'
END_P db ';', 0DH, 0AH, '$'

; П Р О Ц Е Д У Р Ы
;-------------------------------------------------
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
; б а й т в AL п е р е в о д и т с я в д в а с и м в о л а 16-г о ч и с л а 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 с т а р ш а я ц и ф р а
        pop CX           ; в AH м л а д ш а я
        ret
BYTE_TO_HEX ENDP
;-----------------------------
WRD_TO_HEX PROC near
; п е р е в о д в 16 с/с 16-т и р а з р я д н о г о ч и с л а
; в AX- ч и с л о , в DI- а д р е с п о с л е д н е г о с и м в о л а
        push BX
        mov BH, AH
        call BYTE_TO_HEX
        mov [DI], AH
        dec DI
        mov [DI], AL
        dec DI
        mov AL, BH
        call BYTE_TO_HEX
        mov [DI], AH
        dec DI
        mov [DI], AL
        pop BX
        ret
WRD_TO_HEX ENDP
;--------------------------------------
BYTE_TO_DEC PROC near
; п е р е в о д в 10 с/с , в SI- а д р е с п о л я м л а д ш е й ц и ф р ы
        push CX
        push DX
        xor AH, AH
        xor DX, DX
        mov CX, 10
loop_bd: div CX
        or DL, 30h
        mov [SI], DL
        dec SI
        xor DX, DX
        cmp AX, 10
        jae loop_bd
        cmp AL, 00h
        je end_1
        or AL, 30h
        mov [SI], AL
end_1:  pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP
;---------------------
; К О Д
```

```
PRINT PROC near
        mov AH, 09h
        int 21h
        ret
PRINT ENDP

PRINT_SYM PROC near
        push AX
        mov AH, 02h
        int 21h
        pop AX
        ret
PRINT_SYM ENDP

_UNV PROC near
        mov DI, offset UNV
        add DI, 35
        mov AX, DS:[2h]
        call WRD_TO_HEX
        mov DX, offset UNV
        call PRINT
        ret
_UNV ENDP

_ENV PROC near
        mov DI, offset ENV
        add DI, 28
        mov AX, DS:[2Ch]
        call WRD_TO_HEX
        mov DX, offset ENV
        call PRINT
        ret
_ENV ENDP

_CMD PROC near
        mov DX, offset CMD
        call PRINT
        mov CL, DS:[80h]
        cmp CL, 0
        je empty
        mov BX, 81h

lp:
                mov DL, DS:[BX]
                call PRINT_SYM
                inc BX
                loop lp
```

```
        mov DX, offset END_C
        call PRINT
        ret
empty:
        mov DX, offset CMD_Emp
        call PRINT
        ret
_CMD ENDP

_ENV_Cnt_and_PATH PROC near

ENV_Contents:
        mov DX, offset ENV_Cnt
        call PRINT
        mov ES, DS:[2Ch]
        xor DI, DI
        mov DL, ES:[DI]
read_env:
        cmp DL, 0
        je final_env
        call PRINT_SYM
        inc DI
        mov DL, ES:[DI]
        jmp read_env

final_env:
        mov DL, 0Dh
        call PRINT_SYM
        mov DL, 0Ah
        call PRINT_SYM
        inc DI
        mov DL, ES:[DI]
        cmp DL, 00h
        jne read_env

module_PATH:
        mov DX, offset PATH
        call PRINT
        add DI, 2
        mov DL, ES:[DI]
        inc DI

read_path:
        call PRINT_SYM
        mov DL, ES:[DI]
        inc DI
        cmp DL, 0
        jne read_path
```

```
        mov DX, offset END_P
        call PRINT
        ret
_ENV_Cnt_and_PATH ENDP

BEGIN:
        call _UNV
        call _ENV
        call _CMD
        call _ENV_Cnt_and_PATH

        xor AL, AL

        mov AH,01h
        int 21h

        mov AH, 4Ch
        int 21h
TESTPC  ENDS
        END START
```