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 .

Студент гр. 0382 _____ Санников В.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2022

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры.

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.

Ход работы.

Для выполнения лабораторной работы был написан .EXE модуль, который содержит следующие процедуры:

1)FREE_MEMORY — процедура проверки и очистки памяти.

2)IS_LOAD_FILE — процедура загрузки файла и проверки на возможные ошибки при загрузке.

3)PREPARE_PATH — процедура получения пути до вызываемого каталога.

Выполнение пунктов:

1) Был написан и отлажен программный модуль типа .EXE, который выполняет требуемые по заданию функции.

2) Была запущена и отлажена программа, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Был введен символ «c». См. Рисунок 1.

Рисунок 1 — Результат работы .exe модуля при введенном символе «c».

3) Была запущена и отлажена программа, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Вот этот раз вводится комбинация клавиш Ctrl + C. Так как в DOSBOX данная комбинация клавиш никак не обрабатывается, то выводится символ (сердечко). См. Рисунок 2.

Рисунок 2 — Вводится комбинация клавиш Ctrl + C.

4) Была запущена отлаженная программа, когда текущим каталогом является другой каталог, отличный от того, в котором содержатся разработанные программные модули. Вводится символ «c». См. Рисунок 3.

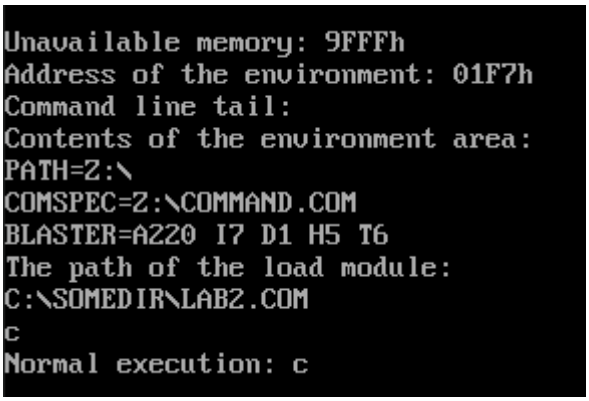

Рисунок 3 — Запуск модуля при другом каталоге.

5) Теперь запустим и отладим программу, когда модули находятся в разных каталогах. См. Рисунок 4.

Рисунок 4 — Запуск программы, когда модули в разных каталогах.

Как видно из данного рисунка, программа выводит ошибку, так как модули в разных каталогах.

Исходный код программы см в приложении А.

Ответы на контрольные вопросы.

1) Как реализовано прерывание Ctrl-C?

Ответ. При нажатии сочетания клавиш Ctrl+C срабатывает прерывание int 23h, управление передается по адресу — (0000:008C), адрес копируется в PSP (с помощью функций 26h и 4ch), при выходе из программы исходное значение адреса восстанавливается.

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

Ответ. В точке вызова функции 4ch прерывания int 21h.

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

Ответ. В данном случае, программа завершится в точке, в который была введена и считана комбинация Ctrl+C.

Вывод.

В ходе данной лабораторной работы была исследована возможность построения загрузочного модуля динамической структуры. Написан .exe модуль, который запускает .com модуль из данного каталога или выводит определенную ошибку.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main2.asm

```
AStack SEGMENT  STACK
          DW 128 DUP(?)
AStack ENDS
;-------------------------------
DATA SEGMENT
     PARAMETR_BLOCK dw 0 ;сегментный адрес среды
                        dd 0 ;сегмент и смещение командной строки
                        dd 0 ;сегмент и смещение FCB
                        dd 0 ;сегмент и смещение второго FCB

     FILE_NAME db 'lab2.com', 0
     PATH_TMP db 128 DUP(0)
     FLAG db 0
     CMD db 1h, 0dh

     KEEP_SS DW 0
     KEEP_SP DW 0
     KEEP_PSP DW 0

     MEMORY_ERROR_7 db 'Control memory block is destroyed!', 0DH,
0AH, '$'
         MEMORY_ERROR_8 db 'Not enough memory for function!', 0DH,
0AH, '$'
        MEMORY_ERROR_9 db 'Invalid memory address!', 0DH, 0AH, '$'

     ERROR_1 db 'Function number not correct!', 0DH, 0AH, '$'
     ERROR_2 db 'File not found!', 0DH, 0AH, '$'
     ERROR_5 db 'Disk crash!', 0DH, 0AH, '$'
     ERROR_8 db 'Low memory size!', 0DH, 0AH, '$'
     ERROR_10 db 'Bad string enviroment!', 0DH, 0AH, '$'
     ERROR_11 db 'Incorrect format!', 0DH, 0AH, '$'

     AH_ERROR_0 db 'Normal execution:        ', 0DH, 0AH, '$'
     AH_ERROR_1 db 'Crtl-Break execution!', 0DH, 0AH, '$'
     AH_ERROR_2 db 'Device execution error!', 0DH, 0AH, '$'
     AH_ERROR_3 db 'Resident error execution!', 0DH, 0AH, '$'

     NEW_STR db 0DH, 0AH, '$'
     DATA_END db 0
DATA ENDS
;-------------------------------
CODE SEGMENT
   ASSUME CS:CODE, DS:DATA, SS:AStack
;-------------------------------
PRINT PROC
     push ax
```

```
      mov ah, 09h
      int 21h
      pop ax
      ret
PRINT ENDP
;------------------------------
FREE_MEMORY PROC
      push ax
      push bx
      push cx
      push dx
      mov ax, offset DATA_END
      mov bx, offset end_main
      add bx, ax
      mov cl, 4
      shr bx, cl
      add bx, 2bh
      mov ah, 4ah
      int 21h
      jnc end_free
      mov FLAG, 1

error_7:
      cmp ax, 7
      jne error_8
      mov dx, offset MEMORY_ERROR_7
      call PRINT
      jmp exit_free

error_8:
      cmp ax, 8
      jne error_9
      mov dx, offset MEMORY_ERROR_8
      call PRINT
      jmp exit_free

error_9:
      cmp ax, 9
      mov dx, offset MEMORY_ERROR_9
      call PRINT
      jmp exit_free

end_free:
      mov flag, 1
      mov dx, offset NEW_STR
      call PRINT

exit_free:
      pop dx
      pop cx
      pop bx
      pop ax
```

```
      ret
FREE_MEMORY ENDP
;------------------------------
LOAD_FILE PROC
      push ax
      push bx
      push cx
      push dx
      push ds
      push es
      mov KEEP_SP, sp
      mov KEEP_SS, ss
      mov ax, DATA
      mov es, ax
      mov bx, offset PARAMETR_BLOCK
      mov dx, offset CMD
      mov [bx+2], dx
      mov [bx+4], ds
      mov dx, offset PATH_TMP
      mov ax, 4b00h
      int 21h
      mov ss, KEEP_SS
      mov sp, KEEP_SP
      pop es
      pop ds
      jnc load_okey

      cmp ax, 1
      jne error_no_file

      mov dx, offset ERROR_1
      call PRINT

      jmp exit_load

err_2:
      cmp ax, 2
      jne err_5
      mov dx, offset ERROR_2
      call PRINT
      jmp exit_load

err_5:
      cmp ax, 5
      jne err_8
      mov dx, offset ERROR_5
      call PRINT
      jmp exit_load

err_8:
      cmp ax, 8
      jne err_10
```

```
      mov dx, offset ERROR_8
      call PRINT
      jmp exit_load

err_10:
      cmp ax, 10
      jne err_11
      mov dx, offset ERROR_10
      call PRINT
      jmp exit_load

err_11:
      cmp ax, 11
      mov dx, offset ERROR_11
      call PRINT
      jmp exit_load

load_okey:
      mov ah, 4dh
      mov al, 00h
      int 21h
      cmp ah, 0
      jne ah_1
      push di
      mov di, offset AH_ERROR_0
      mov [di+18], al
      pop si
      mov dx, offset NEW_STR
      call PRINT
      mov dx, offset AH_ERROR_0
      call PRINT
      jmp exit_load

ah_1:
      cmp ah, 1
      jne ah_2
      mov dx, offset AH_ERROR_1
      call PRINT
      jmp exit_load

ah_2:
      cmp ah, 2
      jne ah_3
      mov dx, offset AH_ERROR_2
      call PRINT
      jmp exit_load

ah_3:
      cmp ah, 3
      mov dx, offset AH_ERROR_3
      call PRINT
```

```
exit_load:
      pop dx
      pop cx
      pop bx
      pop ax
      ret
LOAD_FILE ENDP
;-------------------------------
PREPARE_PATH PROC
      push ax
      push bx
      push cx
      push dx
      push di
      push si
      push es
      mov ax, KEEP_PSP
      mov es, ax
      mov es, es:[2ch]
      mov bx, 0

find_path:
      inc bx
      cmp byte ptr es:[bx-1], 0
      jne find_path
      cmp byte ptr es:[bx+1], 0
      jne find_path
      add bx, 2
      mov di, 0

find_loop:
      mov dl, es:[bx]
      mov byte ptr [PATH_TMP + di], dl
      inc di
      inc bx
      cmp dl, 0
      je end_find_loop
      cmp dl, '\'
      jne find_loop
      mov cx, di
      jmp find_loop

end_find_loop:
      mov di, cx
      mov si, 0

end_p:
      mov dl, byte ptr [FILE_NAME + si]
      mov byte ptr [PATH_TMP + di], dl
      inc di
      inc si
      cmp dl, 0
```

```
      jne end_p
      pop es
      pop si
      pop di
      pop dx
      pop cx
      pop bx
      pop ax
      ret
PREPARE_PATH ENDP
;-------------------------------
MAIN PROC far
      push ds
      xor ax, ax
      push ax
      mov ax, DATA
      mov ds, ax
      mov KEEP_PSP, es
      call FREE_MEMORY
      cmp FLAG, 0
      je exit_main
      call PREPARE_PATH
      call LOAD_FILE

      exit_main:
            xor AL, AL
            mov AH, 4Ch
            int 21h
MAIN ENDP
end_main:
CODE ENDS
END MAIN
```

Файл lab2.asm

```
TESTPC SEGMENT
      ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
      ORG  100H
START:     jmp BEGIN
;-------------------------------
UM db 'Unavailable memory:     h', 0DH, 0AH, '$'
EA db 'Address of the environment:     h', 0DH, 0AH, '$'
CLT db 'Command line tail: ', '$'
EMP db 'Command line tail is empty', 0DH, 0AH, '$'
CEA db 'Contents of the environment area:  ', 0DH, 0AH,'$'
PTH db 'The path of the load module:  ', 0DH, 0AH, '$'
;-------------------------------
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 16-го 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
      ret
BYTE_TO_HEX ENDP
;-------------------------------
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-ти разрядного числа
; в AX - число, в 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
      call BYTE_TO_HEX
      mov [DI], AH
      dec DI
      mov [DI], AL
      pop BX
      ret
WRD_TO_HEX ENDP
;---------------------------------------
BYTE_TO_DEC PROC near
; перевод в 10 с/с, в SI - адрес поля младшей цифры
      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
;----------------------
PRINT PROC near
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
PRINT ENDP
;----------------------
PRINT_SYMB PROC near
      push AX
      mov AH, 02h
      int 21h
      pop AX
      ret
PRINT_SYMB ENDP
;----------------------
UM_FUNC PROC near
      mov AX, DS:[2h]
      mov DI, offset UM
      add DI, 23
      call WRD_TO_HEX
      mov DX, offset UM
      call PRINT
      ret
UM_FUNC ENDP
;---------------------------------
EA_FUNC PROC near
      mov AX, DS:[2Ch]
      mov DI, offset EA
      add DI, 31
      call WRD_TO_HEX
      mov DX, offset EA
      call PRINT
      ret
EA_FUNC ENDP
;---------------------------------
CLT_FUNC PROC near
      xor CX, CX
      mov CL, DS:[80h]
      cmp CL, 0h
      je if_empty
      mov DX, offset CLT
      call PRINT
      mov SI, 81h
loop_clt:
```

```
      mov DL, DS:[SI]
      call PRINT_SYMB
      inc SI
      loop loop_clt

      mov DL, 0Dh
      call PRINT_SYMB
      mov DL, 0Ah
      call PRINT_SYMB
      ret
if_empty:
      mov DX, offset EMP
      call PRINT
      ret
CLT_FUNC ENDP
;-----------------------------------
CEA_FUNC PROC near
      mov DX, offset CEA
      call PRINT
      mov ES, DS:[2Ch]
      xor DI, DI
print1:
      mov DL, ES:[DI]
      cmp DL, 0h
      je print2
      call PRINT_SYMB
      inc DI
      jmp print1
print2:
      mov DL, 0Dh
      call PRINT_SYMB
      mov DL, 0Ah
      call PRINT_SYMB
      inc DI
      mov DL, ES:[DI]
      cmp DL, 0h
      jne print1

      mov DX, offset PTH
      call PRINT
      add DI, 3
print3:
      mov DL, ES:[DI]
      cmp DL, 0h
      je end_print
      call PRINT_SYMB
      inc DI
      jmp print3
end_print:
      mov DL, 0dh
      call PRINT_SYMB
      mov DL, 0ah
```

```
      call PRINT_SYMB
      ret
CEA_FUNC ENDP
;-----------------------------------
BEGIN:
      call UM_FUNC
      call EA_FUNC
      call CLT_FUNC
      call CEA_FUNC
      xor AL, AL
      mov AH, 01h
      int 21h
      mov AH, 4Ch
      int 21h
TESTPC ENDS
      END START
```