# 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. 9382 Русинов Д.А.

Преподаватель Ефремов М.А.

Санкт-Петербург 2021

# 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.

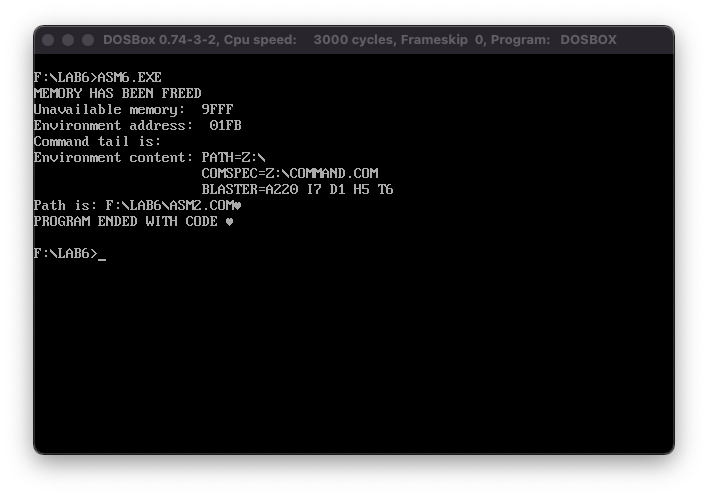
Шаг 5. Запустите отлаженную программу, когда модули находятся в разных каталогах. Занесите полученные данные в отчет.

# Выполнение работы.

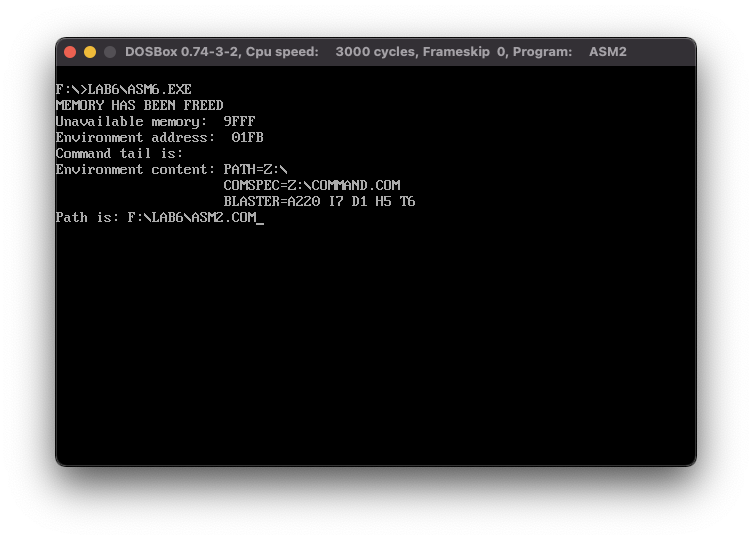
# Была модифицирована программа из лабораторной работы номер два. После введенной модификации, программа перед завершением ожидает символ с клавиатуры. Запустим программу из директории с этими модулями и в конце введем символ 1.

# 

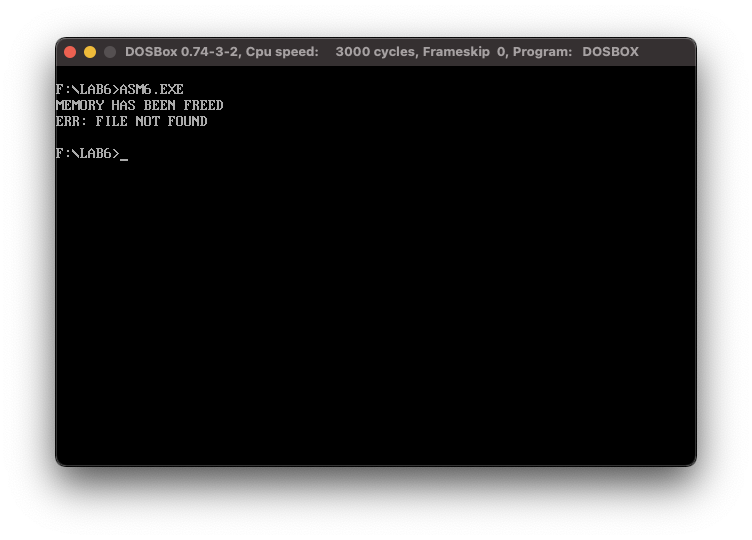
После этого запустим программу и завершим нажатием клавиш Ctrl + C. Программа будет завершена успешно, так как сочетание данных клавиш не реализовано в DOSBox, будет символ “♥”.



Запустим программу при условии, что мы находимся вне директории с этими модулями.



После этого запустим модуль при условии, что модули находятся в разных директориях.



# Вывод.

Был построен загрузочный модуль динамической структуры. Были получены навыки работы с памятью.

# Ответы на контрольные вопросы.

1. Как реализовано прерывание Ctrl-C?

Если было нажато сочетание клавиш Ctrl + C и флаг Break имеет значение ON, то управление будет передано по адресу 0000:008Ch. Затем этот адрес копируется в PSP функциями 26h и 4Ch и восстанавливается при выходе из программы.

1. В какой точке заканчивается вызываемая программа, если код причины завершения 0?

В месте вызова функции 4Ch прерывания 21h.

1. В какой точке заканчивается вызываемая программа по прерыванию Ctrl + C?

В месте вызова функции 01h прерывания 21h.

# ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

STACKS SEGMENT STACK  
 DW 128 DUP(?)  
STACKS ENDS  
  
  
DATA SEGMENT  
 PARAMETER\_BLOCK DW 0  
 DD 0  
 DD 0  
 DD 0   
 PROGRAM DB 'ASM2.COM'**,** 0  
 MEM\_POINTER DB 0  
 SOME\_CMD DB 1H**,** 0DH  
 POS\_CL DB 128 DUP(0)  
 KEEP\_SS DW 0  
 KEEP\_SP DW 0  
 KEEP\_PSP DW 0   
  
 STR\_CRASH\_MCB\_ERR DB 'ERR: MCB CRASHED'**,** 0DH**,** 0AH**,** '$'  
 STR\_NO\_MEM\_ERR DB 'ERR: THERE IS NOT ENOUGH MEMORY TO EXECUTE THIS FUNCTION'**,** 0DH**,** 0AH**,** '$'   
 STR\_SOM\_ADDR\_ERR DB 'ERR: INVALID MEMORY SOM\_ADDRESS'**,** 0DH**,** 0AH**,** '$'  
 STR\_FREE\_MEM DB 'MEMORY HAS BEEN FREED'**,** 0DH**,** 0AH**,** '$'  
  
 STR\_FN\_ERR DB 'ERR: INVALID FUNCTION NUMBER'**,** 0DH**,** 0AH**,** '$'   
 STR\_FILE\_ERROR DB 'ERR: FILE NOT FOUND'**,** 0DH**,** 0AH**,** '$'   
 STR\_DISK\_ERR DB 'ERR: DISK ERROR'**,** 0DH**,** 0AH**,** '$'   
 STR\_MEMORY\_ERROR DB 'ERR: INSUFFICIENT MEMORY'**,** 0DH**,** 0AH**,** '$'   
 STR\_ENVS\_ERR DB 'ERR: WRONG STRING OF ENVIRONMENT '**,** 0DH**,** 0AH**,** '$'   
 STR\_FORMAT\_ERR DB 'ERR: WRONG FORMAT'**,** 0DH**,** 0AH**,** '$'   
   
 STR\_NORM\_FIN DB 0DH**,** 0AH**,** 'PROGRAM ENDED WITH CODE ' **,** 0DH**,** 0AH**,** '$'  
 STR\_CTRL\_END DB 0DH**,** 0AH**,** 'PR ENDED BY CTRL-BREAK' **,** 0DH**,** 0AH**,** '$'  
 STR\_DEVICE\_ERR DB 0DH**,** 0AH**,** 'PR ENDED BY DEVICE ERROR' **,** 0DH**,** 0AH**,** '$'  
 STR\_INT\_END DB 0DH**,** 0AH**,** 'PR ENDED BY INT 31H' **,** 0DH**,** 0AH**,** '$'  
  
 END\_DATA DB 0  
DATA ENDS  
  
  
CODE SEGMENT  
ASSUME CS:CODE**,** DS:DATA**,** SS:STACKS  
  
PRINT\_STRING PROC  
 PUSH AX  
 MOV AH**,** 09H  
 INT 21H   
 POP AX  
 RET  
PRINT\_STRING ENDP  
  
FREE\_MEMORY PROC   
 PUSH AX  
 PUSH BX  
 PUSH CX  
 PUSH DX  
   
 MOV AX**,** OFFSET END\_DATA  
 MOV BX**,** OFFSET EEEND  
 ADD BX**,** AX  
   
 MOV CL**,** 4  
 SHR BX**,** CL  
 ADD BX**,** 2BH  
 MOV AH**,** 4AH  
 INT 21H   
  
 JNC \_END\_SUP\_FREEE  
 MOV MEM\_POINTER**,** 1  
   
CRASH\_MCB:  
 CMP AX**,** 7  
 JNE NOT\_ENOUGH\_MEMORY  
 MOV DX**,** OFFSET STR\_CRASH\_MCB\_ERR  
 CALL PRINT\_STRING  
 JMP SUP\_FREE  
NOT\_ENOUGH\_MEMORY:  
 CMP AX**,** 8  
 JNE SOM\_ADDR  
 MOV DX**,** OFFSET STR\_NO\_MEM\_ERR  
 CALL PRINT\_STRING  
 JMP SUP\_FREE  
SOM\_ADDR:  
 CMP AX**,** 9  
 MOV DX**,** OFFSET STR\_SOM\_ADDR\_ERR  
 CALL PRINT\_STRING  
 JMP SUP\_FREE  
\_END\_SUP\_FREEE:  
 MOV MEM\_POINTER**,** 1  
 MOV DX**,** OFFSET STR\_FREE\_MEM  
 CALL PRINT\_STRING  
   
SUP\_FREE:  
 POP DX  
 POP CX  
 POP BX  
 POP AX  
 RET  
FREE\_MEMORY ENDP  
  
LOAD 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 PARAMETER\_BLOCK  
 MOV DX**,** OFFSET SOME\_CMD  
 MOV [BX+2]**,** DX  
 MOV [BX+4]**,** DS   
 MOV DX**,** OFFSET POS\_CL  
   
 MOV AX**,** 4B00H   
 INT 21H   
   
 MOV SS**,** KEEP\_SS  
 MOV SP**,** KEEP\_SP  
 POP ES  
 POP DS  
   
 JNC LOADS  
   
FN\_ERR:  
 CMP AX**,** 1  
 JNE FILE\_ERR  
 MOV DX**,** OFFSET STR\_FN\_ERR  
 CALL PRINT\_STRING  
 JMP LOAD\_END  
FILE\_ERR:  
 CMP AX**,** 2  
 JNE DISK\_ERR  
 MOV DX**,** OFFSET STR\_FILE\_ERROR  
 CALL PRINT\_STRING  
 JMP LOAD\_END  
DISK\_ERR:  
 CMP AX**,** 5  
 JNE MEM\_ERR  
 MOV DX**,** OFFSET STR\_DISK\_ERR  
 CALL PRINT\_STRING  
 JMP LOAD\_END  
MEM\_ERR:  
 CMP AX**,** 8  
 JNE ENVS\_ERR  
 MOV DX**,** OFFSET STR\_MEMORY\_ERROR  
 CALL PRINT\_STRING  
 JMP LOAD\_END  
ENVS\_ERR:  
 CMP AX**,** 10  
 JNE FORMAT\_ERR  
 MOV DX**,** OFFSET STR\_ENVS\_ERR  
 CALL PRINT\_STRING  
 JMP LOAD\_END  
FORMAT\_ERR:  
 CMP AX**,** 11  
 MOV DX**,** OFFSET STR\_FORMAT\_ERR  
 CALL PRINT\_STRING  
 JMP LOAD\_END  
  
LOADS:  
 MOV AH**,** 4DH  
 MOV AL**,** 00H  
 INT 21H   
   
\_NEND:  
 CMP AH**,** 0  
 JNE CTRLC  
 PUSH DI   
 MOV DI**,** OFFSET STR\_NORM\_FIN  
 MOV [DI+26]**,** AL   
 POP SI  
 MOV DX**,** OFFSET STR\_NORM\_FIN  
 CALL PRINT\_STRING  
 JMP LOAD\_END  
CTRLC:  
 CMP AH**,** 1  
 JNE DEVICE  
 MOV DX**,** OFFSET STR\_CTRL\_END   
 CALL PRINT\_STRING  
 JMP LOAD\_END  
DEVICE:  
 CMP AH**,** 2   
 JNE INT\_31H  
 MOV DX**,** OFFSET STR\_DEVICE\_ERR  
 CALL PRINT\_STRING  
 JMP LOAD\_END  
INT\_31H:  
 CMP AH**,** 3  
 MOV DX**,** OFFSET STR\_INT\_END  
 CALL PRINT\_STRING  
  
LOAD\_END:  
 POP DX  
 POP CX  
 POP BX  
 POP AX  
 RET  
LOAD ENDP  
  
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  
   
FINDZ:  
 INC BX  
 CMP BYTE PTR ES:[BX-1]**,** 0  
 JNE FINDZ  
  
 CMP BYTE PTR ES:[BX+1]**,** 0   
 JNE FINDZ  
   
 ADD BX**,** 2  
 MOV DI**,** 0  
   
\_LOOP:  
 MOV DL**,** ES:[BX]  
 MOV BYTE PTR [POS\_CL+DI]**,** DL  
 INC DI  
 INC BX  
 CMP DL**,** 0  
 JE \_END\_LOOP  
 CMP DL**,** '\'  
 JNE \_LOOP  
 MOV CX**,** DI  
 JMP \_LOOP  
\_END\_LOOP:  
 MOV DI**,** CX  
 MOV SI**,** 0  
   
\_FN:  
 MOV DL**,** BYTE PTR [PROGRAM+SI]  
 MOV BYTE PTR [POS\_CL+DI]**,** DL  
 INC DI   
 INC SI  
 CMP DL**,** 0   
 JNE \_FN  
   
   
 POP ES  
 POP SI  
 POP DI  
 POP DX  
 POP CX  
 POP BX  
 POP AX  
 RET  
PATH ENDP  
  
BEGIN PROC FAR  
 PUSH DS  
 XOR AX**,** AX  
 PUSH AX  
 MOV AX**,** DATA  
 MOV DS**,** AX  
 MOV KEEP\_PSP**,** ES  
 CALL FREE\_MEMORY   
 CMP MEM\_POINTER**,** 0  
 JE \_END  
 CALL PATH  
 CALL LOAD  
\_END:  
 XOR AL**,** AL  
 MOV AH**,** 4CH  
 INT 21H  
   
BEGIN ENDP  
  
EEEND:  
CODE ENDS  
END BEGIN