**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ  
по лабораторной работе № 6**

**по дисциплине «Операционные системы»**

**Тема: «Построение модуля динамической структуры»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8304 |  | Сергеев А.Д. |
| Преподаватель |  | Губкин А.Ф. |

Санкт-Петербург

2020

**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.

**Постановка задачи:**

**Шаг 1.** Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .ЕХЕ, 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.

**Необходимые сведения для составления программы.**

Для загрузки и выполнения одной программы из другой используется функция 4B00h прерывания int 21h (загрузчик ОС). Перед обращением к этой функции необходимо выполнить следующие действия:

1. Подготовить место в памяти. При начальном запуске программы ей отводится вся доступная в данный момент память 0S, поэтому необходимо освободить место в памяти. Для этого можно использовать функцию 4Ah прерывания int 21h. Эта функция позволяет уменьшить отведенный программе блок памяти. Перед вызовом функции надо определить объем памяти, необходимый программе ЛР 6 и задать в регистре BX число параграфов, которые будут выделяться программе. Если функция 4Ah не может быть выполнена, то устанавливается флаг переноса CP=1 и в AX заносится код ошибки:

7 — разрушен управляющий блок памяти;

8 — недостаточно памяти для выполнения функции;

9 — неверный адрес блока памяти;

Поэтому после выполнения каждого прерывания int 21h следует проверять флаг переноса CP=1.

1. Создать блок параметров. Блок параметров - это 14-байтовый блок памяти, в который помещается следующая информация:

dw — сегментный адрес среды

dd — сегмент и смещение командной строки

dd — сегмент и смещение первого FCB

dd — сегмент и смещение второго FCB

Если сегментный адрес среды 0, то вызываемая программа наследует среду вызывающей программы. В противном случае вызывающая программа должна сформировать область памяти в качестве среды, начинающуюся с адреса кратного 16 и поместить этот адрес в блок параметров.

Командная строка записывается в следующем формате:

Первый байт — счетчик, содержащий число символов в командной строке, затем сама командная строка, содержащая не более 128 символов.

На блок параметров перед загрузкой вызываемой программы должны указывать ES:BX.

1. Подготовить строку, содержащую путь и имя вызываемой программы. В конце строки должен стоять код ASCII 0. На подготовленную строку должны указывать DS:DX.
2. Сохранить содержимое регистров SS и SР в переменных. При восстановлении SS и SP нужно учитывать, что DS необходимо также восстановить.

Код завершения формируется вызываемой программой в регистре AL перед выходом в OS с помощью функции 4Ch прерывания int 21h.

В качестве вызываемой программы целесообразно использовать программу, разработанную в Лабораторной работе №2, модифицировав ее следующим образом. Перед выходом из программы перед выполнением функции 4Ch прерывания int 21h следует запросить с клавиатуры символ и поместить введенный символ в регистр AL, в качестве кода завершения. Это можно сделать с помощью функции 01h прерывания int 21h.

Введенный символ остается в регистре AL и служит аргументом для функции 4Ch прерывания int 21h.

**Описание программы.**

В результате выполнения лабораторной работы была написана программа, описание функций которой представлено ниже.

- HEX\_BYTE\_PRINT - выводит на экран один байт в 16 системе счисления, принимая его в регистре AL;

- PRINT\_STRING - вывод строки из DX на экран;

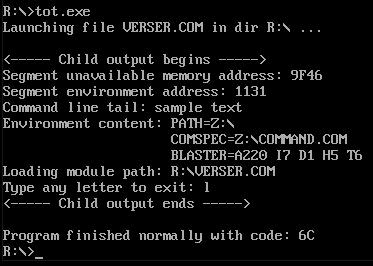
- FREE\_MEM - освобождение незанятого прграммой места в памяти;

- FINISH - вывод сообщения на экран в зависимости от завершения вложенной программы;

- FORM\_PATHS - формирование пути к текущему каталогу и пути к исполняемому модулю;

**Ход работы**

Написание исходного кода производилось в редакторе Atom на базе операционной системы Windows 10, сборка и отладка производились в эмуляторе DOSBox. Поскольку эмуляторы DOS не поддерживают прерывания по Ctrl+Break, отладка обработки этого прерывания проводилась в системе Windows XP.

Рисунок 1 — Выполнение программы с прерыванием по введённой букве «1»

Как видно из рисунка, программа завершилась в штатном режиме с кодом 6C.

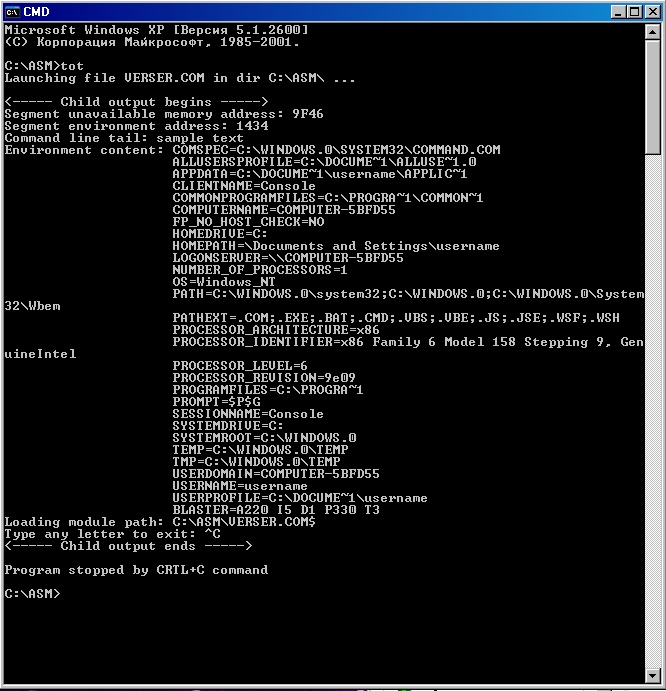
Рисунок 2 – Выполнение программы с прерыванием по введённой комбинации «Ctrl+C»

Рисунок 3 — Программа и модули находятся в разных каталогах

Как видно из рисунка, модуль найден не был, программа verser.exe не была запущена.

**Вывод.**

В результате выполнения данной лабораторной работы была изучена возможность встраивания пользовательского обработчика прерываний от клавиатуры в стандартный.

**Контрольные вопросы.**

**Как реализовано прерывание Ctrl+C:**

Когда нажата комбинация Ctrl+C, DOS вызывает прерывание int 23h. Уровень чувствительности к этому прерыванию может быть проверен и установлен функцией 33h прерывания int 21h.

**В какой точке заканчивается вызываемая программа, если код завершения 0:**

В месте вызова функции 4Ch прерывания int 21h.

**В какой точке заканчивается вызываемая программа по прерыванию Ctrl+Break:**

В точке вызова функции 01h прерывания int 21h, гда ожидался ввод из клавиатуры.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ. TOT.ASM**

ASSUME CS:CODE, DS:DATA, ss:ASTACK

ASTACK SEGMENT STACK

dw 128 dup(0)

ASTACK ENDS

DATA SEGMENT

COMP\_SUCCESS db "Program finished normally with code: $"

COMP\_CC db "Program stopped by CRTL+C command", 13, 10, "$"

ERR\_FILENAME db "File not found!", 13, 10, "Path searched: $"

ERR\_MEMORY db "Not sufficient memory!", 13, 10, "$"

START\_STR\_NAME db "Launching file $"

START\_STR\_PATH db " in dir $"

START\_STR\_END db " ...", 13, 10, "$"

DELIM\_START db 13, 10, "<----- Child output begins ----->", 13, 10, "$"

DELIM\_END db "<----- Child output ends ----->", 13, 10, 13, 10, "$"

PATH db 100 dup(0)

FILENAME db "VERSER.COM$"

MODULE db 150 dup(0)

PSP dw ?

CMD db 12, "sample text", 0

KEEP\_SS dw ?

KEEP\_SP dw ?

PARAMS dw 7 dup(?)

MEMORY\_ERROR db 0

DATA ENDS

CODE SEGMENT

HEX\_BYTE\_PRINT PROC

push ax

push bx

push dx

mov ah, 0

mov bl, 10h

div bl

mov dx, ax

mov ah, 02h

cmp dl, 0Ah

jl PRINT

add dl, 07h

PRINT:

add dl, '0'

int 21h;

mov dl, dh

cmp dl, 0Ah

jl PRINT\_EXT

add dl, 07h

PRINT\_EXT:

add dl, '0'

int 21h;

pop dx

pop bx

pop ax

ret

HEX\_BYTE\_PRINT ENDP

PRINT\_STRING PROC near

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT\_STRING ENDP

FREE\_MEM PROC

mov bx, offset PROGEND

mov ax, es

sub bx, ax

mov cl, 4

shr bx, cl

mov ah, 4Ah

int 21h

jc MCATCH

jmp MDEFAULT

MCATCH:

mov MEMORY\_ERROR, 1

MDEFAULT:

ret

FREE\_MEM ENDP

FINISH PROC

mov ah, 4Dh

int 21h

cmp ah, 1

je ECTRLC

mov dx, offset COMP\_SUCCESS

call PRINT\_STRING

call HEX\_BYTE\_PRINT

mov dl, ah

mov ah, 2h

int 21h

jmp EDEFAULT

ECTRLC:

mov dx, offset COMP\_CC

call PRINT\_STRING

EDEFAULT:

ret

FINISH ENDP

FORM\_PATHS PROC near

push ax

push es

push si

push di

push dx

mov ax, es:[2Ch]

mov es, ax

mov si, 0

SYMBOL:

add si, 1

mov al, es:[si]

mov ah, es:[si+1]

cmp ax, 0

jne SYMBOL

add si, 4

mov di, 0

SYMB:

mov al, es:[si]

cmp al, 0

je DIR\_END

mov PATH[di], al

inc di

add si, 1

jmp SYMB

DIR\_END:

cmp PATH[di - 1], "\"

je APPENSION

mov PATH[di - 1], "$"

dec di

jmp DIR\_END

APPENSION:

mov PATH[di], "$"

mov si, 0

mov di, 0

ADD\_PATH:

cmp PATH[di], "$"

je ADD\_NAME\_PREP

mov al, PATH[di]

mov MODULE[si], al

inc si

inc di

jmp ADD\_PATH

ADD\_NAME\_PREP:

mov di, 0

ADD\_NAME:

cmp FILENAME[di], "$"

je P\_END

mov al, FILENAME[di]

mov MODULE[si], al

inc si

inc di

jmp ADD\_NAME

P\_END:

mov MODULE[si], "$"

pop dx

pop di

pop si

pop es

pop ax

ret

FORM\_PATHS ENDP

MAIN PROC

mov ax, DATA

mov ds, ax

; -----> Forming path to module

call FORM\_PATHS

mov dx, offset START\_STR\_NAME

call PRINT\_STRING

mov dx, offset FILENAME

call PRINT\_STRING

mov dx, offset START\_STR\_PATH

call PRINT\_STRING

mov dx, offset PATH

call PRINT\_STRING

mov dx, offset START\_STR\_END

call PRINT\_STRING

; -----> Freeing memory

call FREE\_MEM

cmp MEMORY\_ERROR, 0

jne PDEFAULT

; -----> Creating Environment

mov PARAMS[0], 0

; -----> Preparing cmd

mov PARAMS[2], offset CMD

mov PARAMS[4], ds

mov dx, offset DELIM\_START

call PRINT\_STRING

push ds

pop es

mov ax, ds

mov es, ax

mov dx, offset MODULE

mov bx, offset PARAMS

mov KEEP\_SS, ss

mov KEEP\_SP, sp

mov ax, 4B00h

int 21h

mov cx, DATA

mov ds, cx

mov ss, KEEP\_SS

mov sp, KEEP\_SP

mov dx, offset DELIM\_END

call PRINT\_STRING

; -----> Printing results

jc NOFILE

jmp SUCCESS

NOFILE:

mov dx, offset ERR\_FILENAME

call PRINT\_STRING

mov dx, offset FILENAME

call PRINT\_STRING

jmp PDEFAULT

SUCCESS:

call FINISH

PDEFAULT:

mov ah, 4Ch

int 21h

MAIN ENDP

PROGEND:

CODE ENDS

end MAIN

**ПРИЛОЖЕНИЕ Б**

**ИСХОДНЫЙ КОД ПРОГРАММЫ. VERSER.COM**

VERSER SEGMENT

ASSUME CS:VERSER, DS:VERSER, ES:NOTHING, SS:NOTHING

org 100H

START: jmp BEGIN

UNAVAILABLE\_ADDRESS db 'Segment unavailable memory address: $'

ENVIRONMENT\_ADDRESS db 'Segment environment address: $'

COMMAND\_LINE\_TAIL db 'Command line tail: $'

ENVIRONMENT\_CONTENT db 'Environment content: $'

LOADING\_PATH db 'Loading module path: $'

END\_STRING db 'Type any letter to exit: $'

TWO\_SYMBOLS\_CONTAINER db ' $'

ENVIRONMENTAL\_LINE db 0dh, 0ah, ' $'

ENDL\_LINE db 0dh, 0ah, '$'

PRINT\_STRING PROC near

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT\_STRING 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\_IN\_HEX PROC near

push cx

push si

mov al, ah

call TETR\_TO\_HEX

xchg al, ah

mov cl, 4

shr al, cl

call TETR\_TO\_HEX

lea si, TWO\_SYMBOLS\_CONTAINER

mov [si], al

mov [si+1], ah

mov dx, offset TWO\_SYMBOLS\_CONTAINER

call PRINT\_STRING

pop si

pop cx

ret

BYTE\_IN\_HEX ENDP

;--------------------------------------------------------------------------------

PRINT\_REG PROC near

push cx

push dx

mov cx, ax

call BYTE\_IN\_HEX

mov al, ch

call BYTE\_IN\_HEX

mov dx, offset ENDL\_LINE

call PRINT\_STRING

pop dx

pop cx

ret

PRINT\_REG ENDP

;--------------------------------------------------------------------------------

PRINT\_TAIL PROC near

push ax

push cx

push si

mov cx, 0

mov cl, ds:[80h]

cmp CL, 0

je PT\_END

mov si, 0

mov ax, 0

CYCLE: mov al, ds:[81h+si]

call PRINT\_BYTE

add si, 1

loop CYCLE

PT\_END: mov dx, offset ENDL\_LINE

call PRINT\_STRING

pop si

pop cx

pop ax

ret

PRINT\_TAIL ENDP

;--------------------------------------------------------------------------------

PRINT\_BYTE PROC near

push ax

push dx

mov dx, 0h

mov dl, al

mov ah, 02h

int 21h

pop dx

pop ax

ret

PRINT\_BYTE ENDP

;--------------------------------------------------------------------------------

PRINT\_ENV PROC near

push ax

push es

push si

push dx

push bx

mov bx, 2Ch

mov es, [bx]

pop bx

mov si, 0

mov al, es:[si]

SYMBOL: cmp al, 0

jne NOLINE

mov dx, offset ENVIRONMENTAL\_LINE

call PRINT\_STRING

jmp LOOPER

NOLINE: call PRINT\_BYTE

LOOPER: add si, 1

mov al, es:[si]

mov ah, es:[si+1]

cmp ax, 0

jne SYMBOL

mov dx, offset ENDL\_LINE

call PRINT\_STRING

call PRINT\_PATH

pop dx

pop si

pop es

pop ax

ret

PRINT\_ENV ENDP

;--------------------------------------------------------------------------------

PRINT\_PATH PROC near

mov dx, offset LOADING\_PATH

call PRINT\_STRING

add si, 4

SYMB: mov al, es:[si]

cmp al, 0

je P\_END

call PRINT\_BYTE

add si, 1

jmp SYMB

P\_END: ret

PRINT\_PATH ENDP

;--------------------------------------------------------------------------------

BEGIN:

push dx

push ax

mov dx, offset UNAVAILABLE\_ADDRESS

call PRINT\_STRING

mov ax, ds:[02h]

call PRINT\_REG

mov dx, offset ENVIRONMENT\_ADDRESS

call PRINT\_STRING

mov ax, ds:[2Ch]

call PRINT\_REG

mov dx, offset COMMAND\_LINE\_TAIL

call PRINT\_STRING

call PRINT\_TAIL

mov dx, offset ENVIRONMENT\_CONTENT

call PRINT\_STRING

call PRINT\_ENV

mov dx, offset ENDL\_LINE

call PRINT\_STRING

mov dx, offset END\_STRING

call PRINT\_STRING

ENDING:

pop ax

pop dx

mov ah, 01h

int 21h

mov dx, offset ENDL\_LINE

call PRINT\_STRING

mov ah, 4ch

int 21h

ret

VERSER ENDS

END START