**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.

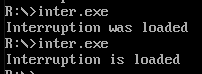
Написанный обработчик заменяет символы “C”, “O” и “D” на “S”, “A” и “T” соответственно. Следует обратить внимание на то, что при установке этого обработчика набрать при помощи клавиатуры название модуля из ЛР 3 (free.com) становится невозможно из-за того, что в нём содержатся заменяемые символы. Для корректной работы это название необходимо заранее скопировать в буфер обмена, а потом вставить, или выполнить модуль до установки прерывания, а дальше использовать навигацию между введёнными командами при помощи стрелочек.

Рисунок 1 — Вывод программы inter.exe после первого запуска

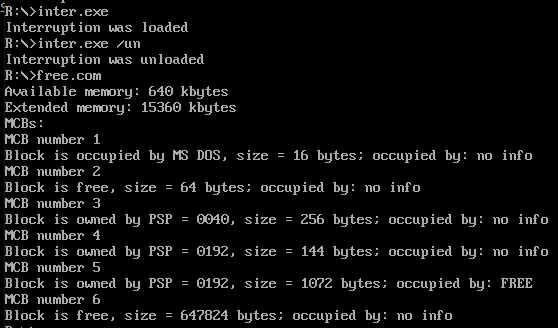
Для тестирования использовалось слово “COLDBLOODED” и фраза “OCEAN WATERS ARE DEEP AND COLD”

Рисунок 2 — Вывод программы free.com после выполнения inter.exe

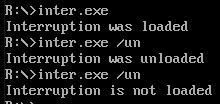
Как видно из рисунка, процедура прерывания осталось резидентной в памяти.

Рисунок 3 — Вывод программы int.exe при повторном запуске

На рисунке 3 показано, что при повторном запуске программа выводит сообщение о том, что резидентный обработчик уже загружен.

Рисунок 4 - Вывод программы free.com после выполнения int.exe с ключом выгрузки

Из рисунка 4 видно, что после выгрузки резидентного обработчика из памяти вся занятая им память была освобождена.

Рисунок 5 — Вывод программы int.exe при повторном запуске с ключом выгрузки

Как видно из рисунка 5, при выгрузке резидентного обработчика было выведено сообщение, а также при запросе повторной выгрузки было показано, что резидентный обработчик не загружен.

**Вывод.**

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

**Контрольные вопросы.**

**Какого типа прерывания использовались в программе:**

В коде использовались программные прерывания, такие, как int 21h, тогда как само обрабатываемое прерывание от клавиатуры (09h) является аппаратным.

**Чем отличается скан-код и ASCII код:**

ASCII код — код символа, необходимый для хранения и печати символа. Скан-код — код клавиши на клавиатуре, необходимый для распознавания нажатых клавиш.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ. INTER.ASM**

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

CODE SEGMENT

INTERRUPT PROC FAR

jmp INT\_START

INT\_DATA:

SUB\_STACK dw 128 dup(0)

INT\_CODE dw 42025

KEEP\_IP dw 0

KEEP\_CS dw 0

KEEP\_SS dw 0

KEEP\_SP dw 0

KEEP\_PSP dw 0

SYMB db 0

TEMP dw 0

INT\_START:

mov KEEP\_SS, ss

mov KEEP\_SP, sp

mov TEMP, seg INTERRUPT

mov ss, TEMP

mov sp, offset SUB\_STACK

add sp, 256

push ax

push bx

push cx

push dx

push si

push es

push ds

mov ax, SEG INTERRUPT

mov ds, ax

in al, 60h

cmp al, 2Eh ; replace 'C'

je INT\_PASS\_S

cmp al, 18h ; replace 'O'

je INT\_PASS\_A

cmp al, 20h ; replace 'D'

je INT\_PASS\_T

pushf

call DWORD PTR KEEP\_IP

jmp INT\_END

INT\_PASS\_S:

mov SYMB, 'S'

jmp INT\_PASS

INT\_PASS\_A:

mov SYMB, 'A'

jmp INT\_PASS

INT\_PASS\_T:

mov SYMB, 'T'

jmp INT\_PASS

INT\_PASS:

in al, 61h

mov ah, al

or al, 80h

out 61h, al

xchg al, al

out 61h, al

mov al, 20h

out 20h, al

INT\_PRINT:

mov ah, 05h

mov cl, SYMB

mov ch, 00h

int 16h

or al, al

jz INT\_END

mov ax, 0040h

mov es, ax

mov ax, es:[1Ah]

mov es:[1Ch], ax

jmp INT\_PRINT

INT\_END:

pop ds

pop es

pop si

pop dx

pop cx

pop bx

pop ax

mov ss, KEEP\_SS

mov sp, KEEP\_SP

mov al, 20h

out 20h, al

IRET

INTERRUPT ENDP

LAST\_BYTE:

LOAD PROC

push ax

push bx

push cx

push dx

push es

push ds

mov ah, 35h

mov al, 09h

int 21h

mov KEEP\_IP, bx

mov KEEP\_CS, es

mov dx, offset INTERRUPT

mov ax, seg INTERRUPT

mov ds, ax

mov ah, 25h

mov al, 09h

int 21h

pop ds

mov dx, offset LAST\_BYTE

add dx, 100h

mov cl, 4h

shr dx, cl

inc dx

mov ah, 31h

int 21h

pop es

pop dx

pop cx

pop bx

pop ax

ret

LOAD ENDP

UNLOAD PROC

push ax

push bx

push dx

push ds

push es

push si

mov ah, 35h

mov al, 09h

int 21h

mov si, offset KEEP\_IP

sub si, offset INTERRUPT

mov dx, es:[bx + si]

mov si, offset KEEP\_CS

sub si, offset INTERRUPT

mov ax, es:[bx + si]

push ds

mov ds, ax

mov ah, 25h

mov al, 09h

int 21h

pop ds

mov si, offset KEEP\_PSP

sub si, offset INTERRUPT

mov ax, es:[bx + si]

mov es, ax

push es

mov ax, es:[2Ch]

mov es, ax

mov ah, 49h

int 21h

pop es

mov ah, 49h

int 21h

pop si

pop es

pop ds

pop dx

pop bx

pop ax

sti

ret

UNLOAD ENDP

INT\_CHECK PROC

push ax

push bx

push si

mov ah, 35h

mov al, 09h

int 21h

mov si, offset INT\_CODE

sub si, offset INTERRUPT

mov ax, es:[bx + si]

cmp ax, INT\_CODE

jne INT\_CHECK\_END

mov INT\_LOADED, 1

INT\_CHECK\_END:

pop si

pop bx

pop ax

ret

INT\_CHECK ENDP

TAIL\_CHECK PROC

cmp byte ptr es:[82h], '/'

jne CL\_CHECK\_END

cmp byte ptr es:[83h], 'u'

jne CL\_CHECK\_END

cmp byte ptr es:[84h], 'n'

jne CL\_CHECK\_END

mov TAIL\_UN, 1

CL\_CHECK\_END:

ret

TAIL\_CHECK ENDP

PRINT\_STRING PROC NEAR

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT\_STRING ENDP

MAIN PROC

push ds

xor ax, ax

push ax

mov ax, DATA

mov ds, ax

mov KEEP\_PSP, es

call TAIL\_CHECK

call INT\_CHECK

cmp TAIL\_UN, 1

je INT\_UNLOAD

cmp INT\_LOADED, 1

jne INT\_LOAD

mov dx, offset IS\_LOADED\_INFO

call PRINT\_STRING

jmp MAIN\_END

INT\_LOAD:

mov dx, offset LOADED\_INFO

call PRINT\_STRING

call LOAD

jmp MAIN\_END

INT\_UNLOAD:

cmp INT\_LOADED, 1

jne NOT\_EXIST

call UNLOAD

mov dx, offset NOT\_LOADED\_INFO

call PRINT\_STRING

jmp MAIN\_END

NOT\_EXIST:

mov dx, offset IS\_NOT\_LOADED\_INFO

call PRINT\_STRING

MAIN\_END:

xor al, al

mov ah, 4Ch

int 21h

MAIN ENDP

CODE ENDS

ASTACK SEGMENT STACK

dw 128 dup(0)

ASTACK ENDS

DATA SEGMENT

LOADED\_INFO db "Interruption was loaded$"

IS\_LOADED\_INFO db "Interruption is loaded$"

NOT\_LOADED\_INFO db "Interruption was unloaded$"

IS\_NOT\_LOADED\_INFO db "Interruption is not loaded$"

INT\_LOADED db 0

TAIL\_UN db 0

DATA ENDS

END MAIN