Московский государственный технический университет

им. Н.Э. Баумана

|  |  |
| --- | --- |
| УТВЕРЖДАЮ: |  |
| Большаков С.А. | "\_17**\_"\_\_\_**апреля\_\_\_\_\_2025г. |

Курсовая работа по курсу «Системное программирование»

**«Резидентная программа (TSR)»**

**Вариант № 21**

Листинг и исходный код программ

(вид документа)

писчая бумага

(вид носителя)

6

(количество листов)

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛИ: |  |
| студент группы ИУ5-43Б | \_\_\_\_\_\_\_\_\_Сорокин М.А.\_\_ |
|  | "17"\_\_апреля\_\_\_\_\_\_\_2025 г. |

Москва – 2025

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc194686650)

[1. Исходный код (KRBUFF) 2](#_Toc194686651)

[2. Листинг Программы (KRBUFF) 15](#_Toc194686652)

[3. Исходный код Программы (UNLOADER) 45](#_Toc194686653)

[4. Листинг Программы (UNLOADER) 45](#_Toc194686654)

# Исходный код (KRBUFF)

; =======================================

**; krbuff.asm**

; 14.04.23

; Сборка:

; > tasm.exe /l krbuff.asm

; > tlink /t /x kr.obj

; =======================================

code segment 'code'

assume CS:code, DS:code

org 100h

\_**start:**

jmp \_initTSR ; на начало программы

ignoredChars DB 'abcde' ; игнорируемые символы

ignoredLength DB 5 ; длина строки ignoredChars

ignoreEnabled DB 0 ; флаг функции игнорирования ввода

translateFrom DB 'QWERTY' ; заменяемые символы

translateTo DB 'ЙЦУКЕН' ; символы, на которые будет происходить замена

translateLength DB 6 ; длина строки translateFrom

translateEnabled DB 0 ; флаг функции перевода

signaturePrintingEnabled DB 0 ; флаг вывода подписи

counter DW 0

printDelay **EQU 2** ; задержка перед выводом "подписи" в секундах

signatureLineLength DW 52 ; длина одной строчки подписи

signatureLine1 DB 179, 'Иванов Иван Иванович ', 179

signatureLine2 DB 179, 'ИУ5-4X ', 179

signatureLine3 DB 179, 'Вариант #0 ', 179

tableTop DB '-', 50 dup ('-'), '¬'

tableBottom DB 'L', 50 dup ('-'), '-'

helpMsg DB '> kr.com [/?] ' , 10, 13

DB ' [/?] - вывод данной справки', 10, 13

DB ' выгрузка резидента из памяти Ctrl+U', 10, 13

DB ' F1 - вывод ФИО и группы по таймеру в центре экрана', 10, 13

DB ' F2 - включение/отключения курсивного вывода русского символа В', 10, 13

DB ' F3 - включение/отключение частичной русификации клавиатуры: QWERTY -> ЙЦУКЕН', 10, 13

DB ' F4 - включение/отключение режима блокировки ввода букв abcde', 10, 13, 0

helpMsgLength EQU $-helpMsg

commandLineResult DB 0

cursiveEnabled DB 0 ; флаг перевода символа в курсив

cursiveSymbol DB 00000000b ; символ, составленный из единиц (его курсивный вариант)

DB 00000000b

DB 00000000b

DB 00111110b

DB 00111111b

DB 00110011b

DB 01100110b

DB 01100110b

DB 01111100b

DB 11000110b

DB 11000110b

DB 11000110b

DB 11111100b

DB 00000000b

DB 00000000b

DB 00000000b

charToCursiveIndex DB 'В' ; символ для замены

savedSymbol DB 16 dup(0FFh) ; переменная для хранения старого символа

old\_int9hOffset DW ? ; адрес старого обработчика int 9h

old\_int9hSegment DW ? ; сегмент старого обработчика int 9h

old\_int1ChOffset DW ? ; адрес старого обработчика int 1Ch

old\_int1ChSegment DW ? ; сегмент старого обработчика int 1Ch

old\_int2FhOffset DW ? ; адрес старого обработчика int 2Fh

old\_int2FhSegment DW ? ; сегмент старого обработчика int 2Fh

installedMsg DB 'Резидент загружен.', 0

alreadyInstalledMsg DB 'Резидент уже был загружен.', 0

notInstalledMsg DB 'Резидент не был загружен.$'

removedMsg DB 'Резидент выгружен из памяти.'

removedMsg\_length EQU $-removedMsg

noRemoveMsg DB 'Не удалось выгрузить резидент'

noRemoveMsg\_length EQU $-noRemoveMsg

true EQU 0FFh ; нужно для удобства использования not с флагами

; 0FFh = 11111111b = инверсия 00000000b

; новый обработчик прерывания **int 9h**

; (работа с клавиатурой)

new\_int9h proc far

push SI AX BX CX DX ES DS ; **сохраняем значения всех, изменяемых** регистров в стеке

push BP ; //////////

push CS ; синхронизируем CS и DS

pop DS

pushf

call dword ptr CS:[old\_int9hOffset] ; вызываем стандартный обработчик прерывания

mov AX, 40h ; 40h - сегмент, где хранятся флаги состояния клавиатуры

mov ES, AX

mov BX, ES:[1Ch] ; адрес хвоста

sub BX, 2h ; сместимся назад к последнему введённому символу

cmp BX, 1Eh ; не вышли ли мы за пределы буфера?

jae \_go

mov BX, 3Ch ; хвост вышел за пределы буфера: значит, последний

; введённый символ находится в конце буфера

\_go:

mov DX, ES:[BX] ; в DX 0 введённый символ

\_test\_Fx: ; проверка **F1-F4**

\_F1:

cmp DH, 3Bh ; **F1(?)**

jne \_F2

not signaturePrintingEnabled ; Флаг печати ФИО

mov ES:[1Ch], BX ; блокировка ввода символа

jmp \_quit

\_F2:

cmp DH, 3Ch ; **F2(?)**

jne \_F3

mov ES:[1Ch], BX ; блокировка ввода символа

not cursiveEnabled ; Установка или сброс курсива

call toggleCursive ; перевод символа в курсив и обратно

; в зависимости от флага cursiveEnabled

jmp \_quit

\_F3:

cmp DH, 3Dh **; F3(?)**

jne \_F4

not translateEnabled ; Включение перевода символов

mov ES:[1Ch], BX ; блокировка ввода символа

jmp \_quit

\_F4:

cmp DH, 3Eh **; F4 (?)**

jne \_translateOrIgnore

not ignoreEnabled ; Включение блокировки символоа

mov ES:[1Ch], BX ; блокировка ввода символа

jmp \_quit

\_translateOrIgnore: ; просто выводим набранный символ на экран

; @@@ следующий блок отвечает за выгрузку по Ctrl-U: @@@

; cmp DL, 15h ; проверяем, что введённый символ - это **[Ctrl+U]**

; jne \_notCtrlU

; mov ES:[1Ch], BX ; блокируем символ **[Ctrl+U]**

; mov AH, 0FFh ; выгрузка

; mov AL, 01h

; int 2Fh

; jmp \_quit

;

; \_notCtrlU:

; @@@ конец блока @@@

cmp ignoreEnabled, true ; включен ли режим блокировки ввода?

jne \_checkTranslate

mov SI, 0 ; да, включен

mov CL, ignoredLength ; количество игнорируемых символов

\_checkIgnored:

cmp DL, ignoredChars[SI] ; проверяем, присутствует ли текущий символ в списке игнорируемых

je \_block

inc SI

loop \_checkIgnored ; зацикливаем ignoredLength раз

jmp \_checkTranslate

; блокируем

\_block:

mov ES:[1Ch], BX ; блокировка ввода символа

; если по варианту нужно не блокировать ввод символа,

; а заменять одни символы другими, замените строку выше строкой

; mov ES:[BX], AX

; на месте AX может быть '\*' для замены всех символов множества ignoredChars на звёздочки

; или, для перевода одних символов в другие - завести массив

; replaceWith DB '...', где перечислить символы, на которые пойдёт замена

; и раскомментировать строки ниже:

; xor AX, AX

; mov AL, replaceWith[SI]

; mov ES:[BX], AX ; замена символа

jmp \_quit

\_checkTranslate:

cmp translateEnabled, true ; включен ли режим перевода?

jne \_quit

mov SI, 0 ; да, включен

mov CL, translateLength ; кол-во символов для перевода

\_checkTranslateLoop:

cmp DL, translateFrom[SI] ; присутствует ли текущий символ в списке для перевода?

je \_translate

inc SI

loop \_checkTranslateLoop ; продолжаем, пока не закончим проверять каждый символ

jmp \_quit

\_translate:

xor AX, AX ; переводим

mov AL, translateTo[SI]

mov ES:[BX], AX ; замена символа

\_quit:

pop BP ; ///////////

pop DS ES DX CX BX AX SI ; **восстанавливаем все регистры**

iret

new\_int9h endp

; в зависимости от флага cursiveEnabled меняет начертание символа на курсив и обратно

; сама смена происходит в процедуре changeFont - здесь же подготавливаются данные

toggleCursive proc

push ES AX ; сохраняем регистры

push CS

pop ES

cmp cursiveEnabled, true ; если флаг равен true,

jne \_restoreSymbol ; выполняем замену символа на курсивный вариант,

; предварительно сохраняя старый символ в savedSymbol

call saveFont

mov CL, charToCursiveIndex

\_shiftTable:

add BP, 16 ; получаем в BP таблицу всех символов. адрес указывает на символ 0

; поэтому нужно совершить сдвиг 16\*X - где X - код символа

loop \_shiftTable

push DS ; пpи savefont смещается pегистp ES

pop AX ; поэтомy пpиходится делать такие махинации, чтобы

push ES ; записать полyченный элемент в savedSymbol

pop DS

push AX ; DS -> AX, ES -> DS, AX -> ES => ES и DS поменялись местами

pop ES ; + сохранение старого значения DS в AX

push AX

mov SI, BP

lea DI, savedSymbol ; сохpаняем в пеpеменнyю savedSymbol таблицу нужного символа

mov CX, 16 ; movsb из DS:SI в ES:DI

rep movsb ; исходные позиции сегментов возвpащены

pop DS ; восстановление DS

mov CX, 1 ; заменим написание символа на кypсив

mov DH, 0

mov DL, charToCursiveIndex

lea BP, cursiveSymbol

call changeFont

jmp \_exitToggleCursive

\_restoreSymbol:

mov CX, 1 ; если флаг равен 0, заменяем курсивный символ на старый вариант

mov DH, 0

mov DL, charToCursiveIndex

lea BP, savedSymbol

call changeFont

\_exitToggleCursive:

pop AX

pop ES

ret

toggleCursive endp

; функция смены начертания символа (курсив/нормальное)

;

; входные данные:

; 1) DL = номер символа для замены

; 2) CX = количество символов заменяемых изображений символов

; (начиная с символа указанного в DX)

; 3) ES:BP = адрес таблицы

;

; описание работы процедуры:

; 1) происходит вызов int 10h (видеосервис)

; с функцией AH = 11h (функции знакогенератора)

; параметр AL = 0 сообщает, что будет заменено изображение

; символа для текущего шрифта.

; в случаях, когда AL = 1 или 2, будет заменено изображение

; только для определенного шрифта (8x14 и 8x8 соответственно)

; 2) параметр BH = 0Eh сообщает, что на определение каждого изображения символа

; расходуется по 14 байт (режим 8x14 бит как раз 14 байт)

; 3) параметр BL = 0 - блок шрифта для загрузки (от 0 до 4)

;

; результат:

; изображение указанного(ых) символа(ов) будет заменено

; на предложенное пользователем.

; изменению подвергнутся все символы, находящиеся на экране:

; таким образом, если изображение заменено, старый вариант нигде уже не проявится

changeFont proc

push AX BX DX

mov AX, 1100h

mov BX, 1000h

int 10h

pop DX BX AX

ret

changeFont endp

; функция сохранения нормального начертания символа

;

; входные данные:

; BH - тип возвращаемой символьной таблицы

; = 0 - таблица из int 1fh

; = 1 - таблица из int 44h

; = 2..5 - таблица из 8x14, 8x8, 8x8 (top), 9x14

; = 6 - 8x16

;

; описание работы процедуры:

; происходит вызов int 10h (видеосервис)

; с функцией AH = 11h (функции знакогенератора)

; параметр AL = 30 - подфункция получения информации о EGA

;

; результат:

; 1) в ES:BP находится таблица символов (полная)

; 2) в CX находится байт на символ

; 3) в DL количество экранных строк

; важно! происходит сдвиг регистра ES (ES = C000h)

saveFont proc

push AX BX DX

mov AX, 1130h

mov BX, 0600h

int 10h

pop BX AX DX

ret

saveFont endp

; обработчик прерывания **int 2Fh**

; служит для:

; 1) проверки факта присутствия TSR в памяти (при AH=0FFh, AL=0)

; будет возвращён AH='i' в случае, если TSR уже загружен

; 2) выгрузки TSR из памяти (при AH=0FFh, AL=1)

new\_int2Fh proc

cmp AH, 0FFh ; наша процедура?

jne \_2Fh\_default ; нет - на стандартный обработчик

cmp AL, 0 ; подпроцедура проверки, загружен ли резидент в память?

je \_alreadyInstalled2Fh

cmp AL, 1 ; подпроцедура выгрузки из памяти?

je \_uninstall

jmp \_2Fh\_default

\_2Fh\_default:

jmp dword ptr CS:[old\_int2FhOffset] ; вызов стандартного обработчика

\_alreadyInstalled2Fh:

mov AH, 'i' ; пусть AH = 'i', если резидент уже загружен в память

iret ; конечно, вместо 'i' может быть любое значение

\_uninstall: ; подпроцедура выгрузки из памяти

push DS ES DX BX

xor BX, BX

push CS ; CS = ES, для доступа к переменным

pop ES

mov AX, 2509h

mov DX, ES:old\_int9hOffset ; возвращаем вектор прерывания 09h на место

mov DS, ES:old\_int9hSegment

int 21h

mov AX, 251Ch

mov DX, ES:old\_int1ChOffset ; возвращаем вектор прерывания 1Ch на место

mov DS, ES:old\_int1ChSegment

int 21h

mov AX, 252Fh

mov DX, ES:old\_int2FhOffset ; возвращаем вектор прерывания 2Fh на место

mov DS, ES:old\_int2FhSegment

int 21h

mov ES, CS:2Ch ; загрузим в ES адрес окружения

mov AH, 49h ; выгрузим из памяти окружение

int 21h

jc \_notRemove

push CS

pop ES ; в ES - адрес резидентной программы

mov AH, 49h ; выгрузим из памяти резидент

int 21h

jc \_notRemove

jmp \_unloaded

\_notRemove: ; не удалось выполнить выгрузку => вывод ошибки

mov AH, 03h ; получаем позицию курсора

int 10h

lea BP, noRemoveMsg

mov CX, noRemoveMsg\_length

mov BL, 0111b

mov AX, 1301h

int 10h

jmp \_2Fh\_exit

\_unloaded: ; выгрузка прошла успешно => вывод сообщения

mov AH, 03h ; получаем позицию курсора

int 10h

lea BP, removedMsg

mov CX, removedMsg\_length

mov BL, 0111b

mov AX, 1301h

int 10h

\_2Fh\_exit:

pop BX DX ES DS

iret

new\_int2Fh endp

; обработчик прерывания **int 1Ch**

; вызывается каждые 55 мс

new\_int1Ch proc far

push AX

push CS

pop DS

pushf

call dword ptr CS:[old\_int1ChOffset] ; вызываем стандартный обработчик прерывания

cmp signaturePrintingEnabled, true ; если нажата управляющая клавиша (в данном случае **F1**)

jne \_notToPrint

cmp counter, printDelay\*1000/55 + 1 ; **если кол-во "тактов" равно printDelay секундам**

je \_letsPrint

jmp \_dontPrint

\_letsPrint:

not signaturePrintingEnabled

mov counter, 0

call printSignature ; выводим подпись на экран

\_dontPrint:

inc counter ; увеличим значение счетчика на 1

\_notToPrint:

pop AX

iret

new\_int1Ch endp

; выводит одну строку подписи

printSignatureLine proc

push DX

mov CX, signatureLineLength

mov BL, 0111b ; цвет выводимого текста

mov AX, 1301h ; AH = 13h - номер ф-ии, AL = 01h - перемещение курсора

int 10h

pop DX

inc DH

ret

printSignatureLine endp

; процедура **вывода подписи**

printSignature proc

push AX DX CX BX ES SP BP SI DI

xor AX, AX ; обнуляем значения регистров

xor BX, BX

xor DX, DX

mov AH, 03h ; чтение текущей позиции курсора

int 10h

push DX ; помещаем информацию о положении курсора в стек

mov DX, 090Fh ; NB! вверху: 000Fh, посередине: 090Fh, внизу: 130Fh

\_actualPrint:

mov AH, 0Fh ; чтение текущего видеорежима. в BH - текущая страница

int 10h

push CS

pop ES ; указываем ES на CS

lea BP, tableTop

call printSignatureLine ; выводим верх таблицы

lea BP, signatureLine1

call printSignatureLine ; выводим первую строку

lea BP, signatureLine2

call printSignatureLine ; выводим вторую строку

lea BP, signatureLine3

call printSignatureLine ; выводим третью строку

lea BP, tableBottom

call printSignatureLine ; выводим низ таблицы

xor BX, BX

pop DX ; восстанавливаем из стека прежнее положение курсора

mov AH, 02h ; меняем положение курсора на первоначальное

int 10h

pop DI SI BP SP ES BX CX DX AX

ret

printSignature endp

; **Основная часть программы**

; 1) установка видеорежима

; 2) проверка, запущен ли резидент

; 3) установка вектора прерываний

\_initTSR:

mov AH, 03h

int 10h

push DX

mov AH, 00h **; установка видеорежима**

mov AL, 83h

int 10h

pop DX

mov AH, 02h

int 10h

call commandParamsParser ; читаем аргументы командной строки

cmp commandLineResult, 2 ; если результат = 2, значит была выведена справка

jne \_shouldContinue ; соответственно, никаких других действий делать не нужно

jmp \_exit

\_shouldContinue:

; @@@ следующий блок нужно оставить, только если нужна выгрузка при перезапуске @@@

; mov AH, 0FFh

; mov AL, 0

; int 2Fh

; cmp AH, 'i' ; **проверка того, загружена ли уже программа**

**; je \_remove**

; @@@ конец блока @@@

; @@@ следующий блок нужно оставить, только если нужна выгрузка при аргументе /u @@@

; cmp commandLineResult, 1 ; проверяем результат работы процедуры

; jne \_go\_on

; mov AH, 0FFh

; mov AL, 0

; int 2Fh

; cmp AH, 'i' ; проверка того, загружена ли уже программа

; je \_remove

;

; mov AH, 09h

; lea DX, notInstalledMsg ; не была загружена

; int 21h

; int 20h

; \_go\_on:

; @@@ конец блока @@@

; @@@ нужно раскомментировать во всех случаях, кроме выгрузки при перезапуске @@@

mov AH, 0FFh ; проверка того, что резидент уже запущен

mov AL, 0

int 2Fh

cmp AH, 'i' ; если запущена, то AH = 'i' (см. процедуру new\_int2Fh)

je \_alreadyInstalled

; @@@ конец блока @@@

mov AX, 3509h ; получить в ES:BX прерывания 09h

int 21h

mov word ptr CS:old\_int9hOffset, BX ; обработчик прерывания 09h

mov word ptr CS:old\_int9hSegment, ES

mov AX, 2509h ; установим вектор на прерывание 09h

mov DX, offset new\_int9h

int 21h

mov AX, 351Ch ; получить в ES:BX прерывания 1Ch

int 21h

mov word ptr CS:old\_int1ChOffset, BX ; обработчик прерывания 1Ch

mov word ptr CS:old\_int1ChSegment, ES

mov AX, 251Ch ; установим вектор на прерывание 1Ch

mov DX, offset new\_int1Ch

int 21h

mov AX, 352Fh ; **получить в ES:BX прерывания 2Fh**

**int 21h**

mov word ptr CS:old\_int2FhOffset, BX ; обработчик прерывания 2Fh

mov word ptr CS:old\_int2FhSegment, ES

mov AX, 252Fh ; **установим вектор на прерывание 2Fh**

mov DX, offset new\_int2Fh

int 21h

lea BX, installedMsg ; выводим сообщение, что всё ОК

call printStr

mov DX, offset \_initTSR ; остаемся в памяти и выходим из основной части

int 27h

\_remove: ; выгрузка из памяти

push ES

mov AX, DS:[2Ch] ; PSP

mov ES, AX

mov AH, 49h ; хватит памяти чтоб остаться резидентом?

int 21h

pop ES

mov AH, 0FFh

mov AL, 1

int 2Fh

jmp \_exit

\_alreadyInstalled: ; резидент уже запущен

lea BX, alreadyInstalledMsg

call printStr

jmp \_exit

\_exit: ; выход

int 20h

; **парсер аргментов командной строки. выводит справку**.

; устанавливает флаг commandLineResult:

; 0 = всё ОК; 1 = нужна выгрузка; 2 = была выведена справка, не нужно загружать резидент

commandParamsParser proc

push CS

pop ES

mov SI, 80h ; SI = смещение командной строки

lodsb ; получим кол-во символов

or AL, AL ; если 0 символов введено,

jz \_paramParsingEnd ; то все в порядке

\_nextChar:

inc SI ; теперь SI указывает на первый символ строки

cmp [SI], BYTE ptr 0

je \_paramParsingEnd

lodsw ; получаем два символа

cmp AX, '?/'

je \_displayHelp

; @@@ следует раскомментировать, если нужно выгружать по аргументу /u @@@

cmp AX, 'u/'

je \_finishTSR

cmp AX, 'U/'

je \_finishTSR

jmp \_nextChar

\_finishTSR:

mov commandLineResult, 1 ; флаг того, что необходимо выгрузить резидент

jmp \_nextChar

; @@@ конец блока @@@

jmp \_paramParsingEnd

\_displayHelp:

lea BX, helpMsg ; выводим справку

call printStr

mov commandLineResult, 2 ; флаг того, что резидент загружать не надо

\_paramParsingEnd:

ret

commandParamsParser endp

; отображает символ из AL

printChar proc

mov AH, 0EH

int 010H

ret

printChar endp

; отображает нуль-**терминированную строку из [BX]**

printStr proc

push DX AX

mov AX, [BX]

\_printStrLoop:

cmp AL, 0

je \_printStrEnd

call printChar

inc BX

mov AX, [BX]

jmp \_printStrLoop

\_printStrEnd:

pop AX DX

ret

printStr endp

code ends

end \_start

# Листинг Программы (KRBUFF)

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 1

krbuff.asm

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 1

krbuff.asm

1 ; =======================================

2 ; **krbuff.asm**

3 ;

4 ; Сборка:

5 ; > tasm.exe /l kr.asm

6 ; > tlink /t /x kr.obj

7 ; =======================================

8

9 0000 code segment 'code'

10 assume CS:code, DS:code

11 org 100h

12

13 0100 \_start:

14 0100 E9 0571 jmp \_initTSR **; на начало** программы

15

16 0103 61 62 63 64 65 ignoredChars DB 'abcde' ; игнорируемые символы

17 0108 05 ignoredLength DB 5 ; длина строки ignoredChars

18 0109 00 ignoreEnabled DB 0 ; флаг функции игнорирования ввода

19 010A 51 57 45 52 54 59 translateFrom DB 'QWERTY' ; заменяемые символы

20 0110 89 96 93 8A 85 8D translateTo DB 'ЙЦУКЕН' ; символы, на которые будет происходить замена

21 0116 06 translateLength DB 6 ; длина строки translateFrom

22 0117 00 translateEnabled DB 0 ; флаг функции перевода

23

24 0118 00 signaturePrintingEnabled DB 0 ; флаг вывода подписи

25 0119 0000 counter DW 0

26 =0002 printDelay EQU 2 ; задержка перед выводом "подписи" в секундах

27

28 011B 0034 signatureLineLength DW 52 ; длина одной строчки подписи

29 011D B3 88 A2 A0 AD AE A2+ signatureLine1 DB 179, 'Иванов Иван Иванович ', +

30 20 88 A2 A0 AD 20 88+ 179

31 A2 A0 AD AE A2 A8 E7+

32 20 20 20 20 20 20 20+

33 20 20 20 20 20 20 20+

34 20 20 20 20 20 20 20+

35 20 20 20 20 20 20 20+

36 20 20 B3

37 0151 B3 88 93 35 2D 34 58+ signatureLine2 DB 179, 'ИУ5-4X ', +

38 20 20 20 20 20 20 20+ 179

39 20 20 20 20 20 20 20+

40 20 20 20 20 20 20 20+

41 20 20 20 20 20 20 20+

42 20 20 20 20 20 20 20+

43 20 20 20 20 20 20 20+

44 20 20 B3

45 0185 B3 82 A0 E0 A8 A0 AD+ signatureLine3 DB 179, 'Вариант #0 ', +

46 E2 20 23 30 20 20 20+ 179

47 20 20 20 20 20 20 20+

48 20 20 20 20 20 20 20+

49 20 20 20 20 20 20 20+

50 20 20 20 20 20 20 20+

51 20 20 20 20 20 20 20+

52 20 20 B3

53 01B9 DA 32\*(C4) BF tableTop DB '┌', 50 dup ('─'), '┐'

54 01ED C0 32\*(C4) D9 tableBottom DB '└', 50 dup ('─'), '┘'

55

56 0221 3E 20 6B 72 2E 63 6F+ helpMsg DB '> kr.com [/?] [/u]', 10, 13

57 6D 20 5B 2F 3F 5D 20+

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 2

krbuff.asm

58 5B 2F 75 5D 0A 0D

59 0235 20 5B 2F 3F 5D 20 2D+ DB ' [/?] - вывод данной справки', 10, 13

60 20 A2 EB A2 AE A4 20+

61 A4 A0 AD AD AE A9 20+

62 E1 AF E0 A0 A2 AA A8+

63 0A 0D

64 0253 20 5B 2F 75 5D 20 2D+ DB ' [/u] - выгрузка резидента из памяти', 10, 13

65 20 A2 EB A3 E0 E3 A7+

66 AA A0 20 E0 A5 A7 A8+

67 A4 A5 AD E2 A0 20 A8+

68 A7 20 AF A0 AC EF E2+

69 A8 0A 0D

70 0279 20 20 46 31 20 20 2D+ DB ' F1 - вывод ФИО и группы по таймеру в центре экрана', 10, 13

71 20 A2 EB A2 AE A4 20+

72 94 88 8E 20 A8 20 A3+

73 E0 E3 AF AF EB 20 AF+

74 AE 20 E2 A0 A9 AC A5+

75 E0 E3 20 A2 20 E6 A5+

76 AD E2 E0 A5 20 ED AA+

77 E0 A0 AD A0 0A 0D

78 02B0 20 20 46 32 20 20 2D+ DB ' F2 - включение/отключения курсивного вывода русского символа В', 10, 13

79 20 A2 AA AB EE E7 A5+

80 AD A8 A5 2F AE E2 AA+

81 AB EE E7 A5 AD A8 EF+

82 20 AA E3 E0 E1 A8 A2+

83 AD AE A3 AE 20 A2 EB+

84 A2 AE A4 A0 20 E0 E3+

85 E1 E1 AA AE A3 AE 20+

86 E1 A8 AC A2 AE AB A0+

87 20 82 0A 0D

88 02F3 20 20 46 33 20 20 2D+ DB ' F3 - включение/отключение частичной русификации клавиатуры: QWERTY -> +

89 20 A2 AA AB EE E7 A5+ ЙЦУКЕН', 10, 13

90 AD A8 A5 2F AE E2 AA+

91 AB EE E7 A5 AD A8 A5+

92 20 E7 A0 E1 E2 A8 E7+

93 AD AE A9 20 E0 E3 E1+

94 A8 E4 A8 AA A0 E6 A8+

95 A8 20 AA AB A0 A2 A8+

96 A0 E2 E3 E0 EB 3A 20+

97 51 57 45 52 54 59 20+

98 2D 3E 20 89 96 93 8A+

99 85 8D 0A 0D

100 0344 20 20 46 34 20 20 2D+ DB ' F4 - включение/отключение режима блокировки ввода букв abcde', 10, 13, 0

101 20 A2 AA AB EE E7 A5+

102 AD A8 A5 2F AE E2 AA+

103 AB EE E7 A5 AD A8 A5+

104 20 E0 A5 A6 A8 AC A0+

105 20 A1 AB AE AA A8 E0+

106 AE A2 AA A8 20 A2 A2+

107 AE A4 A0 20 A1 E3 AA+

108 A2 20 61 62 63 64 65+

109 0A 0D 00

110

111 =0165 helpMsgLength EQU $-helpMsg

112 0386 00 commandLineResult DB 0

113

114 0387 00 cursiveEnabled DB 0 ; флаг перевода символа в курсив

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 3

krbuff.asm

115 0388 00 cursiveSymbol DB 00000000b ; символ, составленный из единиц (его курсивный+

116 вариант)

117 0389 00 DB 00000000b

118 038A 00 DB 00000000b

119 038B 3E DB 00111110b

120 038C 3F DB 00111111b

121 038D 33 DB 00110011b

122 038E 66 DB 01100110b

123 038F 66 DB 01100110b

124 0390 7C DB 01111100b

125 0391 C6 DB 11000110b

126 0392 C6 DB 11000110b

127 0393 C6 DB 11000110b

128 0394 FC DB 11111100b

129 0395 00 DB 00000000b

130 0396 00 DB 00000000b

131 0397 00 DB 00000000b

132

133 0398 82 charToCursiveIndex DB 'В' ; символ для замены

134 0399 10\*(FF) savedSymbol DB 16 dup(0FFh) ; переменная для хранения старого символа

135

136 03A9 ???? old\_int9hOffset DW ? ; адрес старого обработчика int 9h

137 03AB ???? old\_int9hSegment DW ? ; сегмент старого обработчика int 9h

138 03AD ???? old\_int1ChOffset DW ? ; адрес старого обработчика int 1Ch

139 03AF ???? old\_int1ChSegment DW ? ; сегмент старого обработчика int 1Ch

140 03B1 ???? old\_int2FhOffset DW ? ; адрес старого обработчика int 2Fh

141 03B3 ???? old\_int2FhSegment DW ? ; сегмент старого обработчика int 2Fh

142

143 03B5 90 A5 A7 A8 A4 A5 AD+ installedMsg DB 'Резидент загружен.', 0

144 E2 20 A7 A0 A3 E0 E3+

145 A6 A5 AD 2E 00

146 03C8 90 A5 A7 A8 A4 A5 AD+ alreadyInstalledMsg DB 'Резидент уже был загружен.', 0

147 E2 20 E3 A6 A5 20 A1+

148 EB AB 20 A7 A0 A3 E0+

149 E3 A6 A5 AD 2E 00

150 03E3 90 A5 A7 A8 A4 A5 AD+ notInstalledMsg DB 'Резидент не был загружен.$'

151 E2 20 AD A5 20 A1 EB+

152 AB 20 A7 A0 A3 E0 E3+

153 A6 A5 AD 2E 24

154

155 03FD 90 A5 A7 A8 A4 A5 AD+ removedMsg DB 'Резидент выгружен из памяти.'

156 E2 20 A2 EB A3 E0 E3+

157 A6 A5 AD 20 A8 A7 20+

158 AF A0 AC EF E2 A8 2E

159 =001C removedMsg\_length EQU $-removedMsg

160

161 0419 8D A5 20 E3 A4 A0 AB+ noRemoveMsg DB 'Не удалось выгрузить резидент'

162 AE E1 EC 20 A2 EB A3+

163 E0 E3 A7 A8 E2 EC 20+

164 E0 A5 A7 A8 A4 A5 AD+

165 E2

166 =001D noRemoveMsg\_length EQU $-noRemoveMsg

167

168 =00FF true EQU 0FFh ; нужно для удобства использования not с флагами

169 ; 0FFh = 11111111b = инверсия 00000000b

170

171 ; новый обработчик прерывания int 9h

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 4

krbuff.asm

172 ; (работа с клавиатурой)

173 0436 new\_int9h proc far

174 0436 56 50 53 51 52 06 1E push SI AX BX CX DX ES DS ; сохраняем значения всех, изменяемых регистров+

175 в стеке

176 043D 55 push BP ; //////////

177 043E 0E push CS ; синхронизируем CS и DS

178 043F 1F pop DS

179

180 0440 9C pushf

181 0441 2E: FF 1E 03A9r call dword ptr CS:[old\_int9hOffset] ; вызываем стандартный обработчик прерывания

182 0446 B8 0040 mov AX, 40h ; 40h - сегмент, где хранятся флаги состояния +

183 клавиатуры

184 0449 8E C0 mov ES, AX

185 044B 26: 8B 1E 001C mov BX, ES:[1Ch] ; адрес хвоста

186 0450 83 EB 02 sub BX, 2h ; сместимся назад к последнему введённому +

187 символу

188 0453 83 FB 1E cmp BX, 1Eh ; не вышли ли мы за пределы буфера?

189 0456 73 03 jae \_go

190 0458 BB 003C mov BX, 3Ch ; хвост вышел за пределы буфера: значит, +

191 последний

192 ; введённый символ находится в конце буфера

193 045B \_go:

194 045B 26: 8B 17 mov DX, ES:[BX] ; в DX 0 введённый символ

195

196 045E \_test\_Fx: ; проверка F1-F4

197 045E \_F1:

198 045E 80 FE 3B cmp DH, 3Bh ; F1

199 0461 75 0C jne \_F2

200 0463 F6 16 0118r not signaturePrintingEnabled

201 0467 26: 89 1E 001C mov ES:[1Ch], BX ; блокировка ввода символа

202 046C EB 7C 90 jmp \_quit

203 046F \_F2:

204 046F 80 FE 3C cmp DH, 3Ch ; F2

205 0472 75 0F jne \_F3

206 0474 26: 89 1E 001C mov ES:[1Ch], BX ; блокировка ввода символа

207 0479 F6 16 0387r not cursiveEnabled

208 047D E8 0073 call toggleCursive ; перевод символа в курсив и обратно

209 ; в зависимости от флага cursiveEnabled

210 0480 EB 68 90 jmp \_quit

211 0483 \_F3:

212 0483 80 FE 3D cmp DH, 3Dh ; F3

213 0486 75 0C jne \_F4

214 0488 F6 16 0117r not translateEnabled

215 048C 26: 89 1E 001C mov ES:[1Ch], BX ; блокировка ввода символа

216 0491 EB 57 90 jmp \_quit

217 0494 \_F4:

218 0494 80 FE 3E cmp DH, 3Eh ; F4

219 0497 75 0C jne \_translateOrIgnore

220 0499 F6 16 0109r not ignoreEnabled

221 049D 26: 89 1E 001C mov ES:[1Ch], BX ; блокировка ввода символа

222 04A2 EB 46 90 jmp \_quit

223

224 04A5 \_translateOrIgnore: ; просто выводим набранный символ на экран

225 ; @@@ следующий блок отвечает за выгрузку по Ctrl-U: @@@

226 ; cmp DL, 15h ; проверяем, что введённый символ - это [Ctrl+U]

227 ; jne \_notCtrlU

228 ; mov ES:[1Ch], BX ; блокируем символ [Ctrl+U]

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 5

krbuff.asm

229 ; mov AH, 0FFh ; выгрузка

230 ; mov AL, 01h

231 ; int 2Fh

232 ; jmp \_quit

233 ;

234 ; \_notCtrlU:

235 ; @@@ конец блока @@@

236 04A5 80 3E 0109r FF cmp ignoreEnabled, true ; включен ли режим блокировки ввода?

237 04AA 75 1B jne \_checkTranslate

238

239 04AC BE 0000 mov SI, 0 ; да, включен

240 04AF 8A 0E 0108r mov CL, ignoredLength ; количество игнорируемых символов

241

242 04B3 \_checkIgnored:

243 04B3 3A 94 0103r cmp DL, ignoredChars[SI] ; проверяем, присутствует ли текущий символ в +

244 списке игнорируемых

245 04B7 74 06 je \_block

246 04B9 46 inc SI

247 04BA E2 F7 loop \_checkIgnored ; зацикливаем ignoredLength раз

248 04BC EB 09 90 jmp \_checkTranslate

249

250 ; блокируем

251 04BF \_block:

252 04BF 26: 89 1E 001C mov ES:[1Ch], BX ; блокировка ввода символа

253 ; если по варианту нужно не блокировать ввод символа,

254 ; а заменять одни символы другими, замените строку выше строкой

255 ; mov ES:[BX], AX

256 ; на месте AX может быть '\*' для замены всех символов множества ignoredChars на +

257 звёздочки

258 ; или, для перевода одних символов в другие - завести массив

259 ; replaceWith DB '...', где перечислить символы, на которые пойдёт замена

260 ; и раскомментировать строки ниже:

261 ; xor AX, AX

262 ; mov AL, replaceWith[SI]

263 ; mov ES:[BX], AX ; замена символа

264 04C4 EB 24 90 jmp \_quit

265

266 04C7 \_checkTranslate:

267 04C7 80 3E 0117r FF cmp translateEnabled, true ; включен ли режим перевода?

268 04CC 75 1C jne \_quit

269

270 04CE BE 0000 mov SI, 0 ; да, включен

271 04D1 8A 0E 0116r mov CL, translateLength ; кол-во символов для перевода

272

273 04D5 \_checkTranslateLoop:

274 04D5 3A 94 010Ar cmp DL, translateFrom[SI] ; присутствует ли текущий символ в списке для +

275 перевода?

276 04D9 74 06 je \_translate

277 04DB 46 inc SI

278 04DC E2 F7 loop \_checkTranslateLoop ; продолжаем, пока не закончим проверять каждый+

279 символ

280 04DE EB 0A 90 jmp \_quit

281

282 04E1 \_translate:

283 04E1 33 C0 xor AX, AX ; переводим

284 04E3 8A 84 0110r mov AL, translateTo[SI]

285 04E7 26: 89 07 mov ES:[BX], AX ; замена символа

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 6

krbuff.asm

286

287 04EA \_quit:

288 04EA 5D pop BP ; ///////////

289 04EB 1F 07 5A 59 5B 58 5E pop DS ES DX CX BX AX SI ; восстанавливаем все регистры

290 04F2 CF iret

291 04F3 new\_int9h endp

292

293 ; в зависимости от флага cursiveEnabled меняет начертание символа на курсив и обратно

294 ; сама смена происходит в процедуре changeFont - здесь же подготавливаются данные

295 04F3 toggleCursive proc

296 04F3 06 50 push ES AX ; сохраняем регистры

297 04F5 0E push CS

298 04F6 07 pop ES

299

300 04F7 80 3E 0387r FF cmp cursiveEnabled, true ; если флаг равен true,

301 04FC 75 30 jne \_restoreSymbol ; выполняем замену символа на курсивный вариант,

302 ; предварительно сохраняя старый символ в +

303 savedSymbol

304

305 04FE E8 004E call saveFont

306 0501 8A 0E 0398r mov CL, charToCursiveIndex

307 0505 \_shiftTable:

308 0505 83 C5 10 add BP, 16 ; получаем в BP таблицу всех символов. адрес +

309 указывает на символ 0

310 ; поэтому нужно совершить сдвиг 16\*X - где X - +

311 код символа

312 0508 E2 FB loop \_shiftTable

313

314 050A 1E push DS ; пpи savefont смещается pегистp ES

315 050B 58 pop AX ; поэтомy пpиходится делать такие махинации, +

316 чтобы

317 050C 06 push ES ; записать полyченный элемент в savedSymbol

318 050D 1F pop DS

319 050E 50 push AX ; DS -> AX, ES -> DS, AX -> ES => ES и DS +

320 поменялись местами

321 050F 07 pop ES ; + сохранение старого значения DS в AX

322 0510 50 push AX

323

324 0511 8B F5 mov SI, BP

325 0513 BF 0399r lea DI, savedSymbol ; сохpаняем в пеpеменнyю savedSymbol таблицу +

326 нужного символа

327

328 0516 B9 0010 mov CX, 16 ; movsb из DS:SI в ES:DI

329

330 0519 F3> A4 rep movsb ; исходные позиции сегментов возвpащены

331 051B 1F pop DS ; восстановление DS

332

333 051C B9 0001 mov CX, 1 ; заменим написание символа на кypсив

334 051F B6 00 mov DH, 0

335 0521 8A 16 0398r mov DL, charToCursiveIndex

336 0525 BD 0388r lea BP, cursiveSymbol

337 0528 E8 0015 call changeFont

338 052B EB 10 90 jmp \_exitToggleCursive

339

340 052E \_restoreSymbol:

341 052E B9 0001 mov CX, 1 ; если флаг равен 0, заменяем курсивный символ +

342 на старый вариант

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 7

krbuff.asm

343 0531 B6 00 mov DH, 0

344 0533 8A 16 0398r mov DL, charToCursiveIndex

345 0537 BD 0399r lea BP, savedSymbol

346 053A E8 0003 call changeFont

347

348 053D \_exitToggleCursive:

349 053D 58 pop AX

350 053E 07 pop ES

351 053F C3 ret

352 0540 toggleCursive endp

353

354 ; функция смены начертания символа (курсив/нормальное)

355 ;

356 ; входные данные:

357 ; 1) DL = номер символа для замены

358 ; 2) CX = количество символов заменяемых изображений символов

359 ; (начиная с символа указанного в DX)

360 ; 3) ES:BP = адрес таблицы

361 ;

362 ; описание работы процедуры:

363 ; 1) происходит вызов int 10h (видеосервис)

364 ; с функцией AH = 11h (функции знакогенератора)

365 ; параметр AL = 0 сообщает, что будет заменено изображение

366 ; символа для текущего шрифта.

367 ; в случаях, когда AL = 1 или 2, будет заменено изображение

368 ; только для определенного шрифта (8x14 и 8x8 соответственно)

369 ; 2) параметр BH = 0Eh сообщает, что на определение каждого изображения символа

370 ; расходуется по 14 байт (режим 8x14 бит как раз 14 байт)

371 ; 3) параметр BL = 0 - блок шрифта для загрузки (от 0 до 4)

372 ;

373 ; результат:

374 ; изображение указанного(ых) символа(ов) будет заменено

375 ; на предложенное пользователем.

376 ; изменению подвергнутся все символы, находящиеся на экране:

377 ; таким образом, если изображение заменено, старый вариант нигде уже не проявится

378 0540 changeFont proc

379 0540 50 53 52 push AX BX DX

380 0543 B8 1100 mov AX, 1100h

381 0546 BB 1000 mov BX, 1000h

382 0549 CD 10 int 10h

383 054B 5A 5B 58 pop DX BX AX

384 054E C3 ret

385 054F changeFont endp

386

387 ; функция сохранения нормального начертания символа

388 ;

389 ; входные данные:

390 ; BH - тип возвращаемой символьной таблицы

391 ; = 0 - таблица из int 1fh

392 ; = 1 - таблица из int 44h

393 ; = 2..5 - таблица из 8x14, 8x8, 8x8 (top), 9x14

394 ; = 6 - 8x16

395 ;

396 ; описание работы процедуры:

397 ; происходит вызов int 10h (видеосервис)

398 ; с функцией AH = 11h (функции знакогенератора)

399 ; параметр AL = 30 - подфункция получения информации о EGA

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 8

krbuff.asm

400 ;

401 ; результат:

402 ; 1) в ES:BP находится таблица символов (полная)

403 ; 2) в CX находится байт на символ

404 ; 3) в DL количество экранных строк

405 ; важно! происходит сдвиг регистра ES (ES = C000h)

406 054F saveFont proc

407 054F 50 53 52 push AX BX DX

408 0552 B8 1130 mov AX, 1130h

409 0555 BB 0600 mov BX, 0600h

410 0558 CD 10 int 10h

411 055A 5B 58 5A pop BX AX DX

412 055D C3 ret

413 055E saveFont endp

414

415 ; обработчик прерывания int 2Fh

416 ; служит для:

417 ; 1) проверки факта присутствия TSR в памяти (при AH=0FFh, AL=0)

418 ; будет возвращён AH='i' в случае, если TSR уже загружен

419 ; 2) выгрузки TSR из памяти (при AH=0FFh, AL=1)

420 055E new\_int2Fh proc

421 055E 80 FC FF cmp AH, 0FFh ; наша процедура?

422 0561 75 0B jne \_2Fh\_default ; нет - на стандартный обработчик

423 0563 3C 00 cmp AL, 0 ; подпроцедура проверки, загружен ли резидент в+

424 память?

425 0565 74 0C je \_alreadyInstalled2Fh

426 0567 3C 01 cmp AL, 1 ; подпроцедура выгрузки из памяти?

427 0569 74 0B je \_uninstall

428 056B EB 01 90 jmp \_2Fh\_default

429

430 056E \_2Fh\_default:

431 056E 2E: FF 2E 03B1r jmp dword ptr CS:[old\_int2FhOffset] ; вызов стандартного обработчика

432

433 0573 \_alreadyInstalled2Fh:

434 0573 B4 69 mov AH, 'i' ; пусть AH = 'i', если резидент уже загружен в +

435 память

436 0575 CF iret ; конечно, вместо 'i' может быть любое значение

437

438 0576 \_uninstall: ; подпроцедура выгрузки из памяти

439 0576 1E 06 52 53 push DS ES DX BX

440 057A 33 DB xor BX, BX

441

442 057C 0E push CS ; CS = ES, для доступа к переменным

443 057D 07 pop ES

444

445 057E B8 2509 mov AX, 2509h

446 0581 26: 8B 16 03A9r mov DX, ES:old\_int9hOffset ; возвращаем вектор прерывания 09h на место

447 0586 26: 8E 1E 03ABr mov DS, ES:old\_int9hSegment

448 058B CD 21 int 21h

449

450 058D B8 251C mov AX, 251Ch

451 0590 26: 8B 16 03ADr mov DX, ES:old\_int1ChOffset ; возвращаем вектор прерывания 1Ch на место

452 0595 26: 8E 1E 03AFr mov DS, ES:old\_int1ChSegment

453 059A CD 21 int 21h

454

455 059C B8 252F mov AX, 252Fh

456 059F 26: 8B 16 03B1r mov DX, ES:old\_int2FhOffset ; возвращаем вектор прерывания 2Fh на место

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 9

krbuff.asm

457 05A4 26: 8E 1E 03B3r mov DS, ES:old\_int2FhSegment

458 05A9 CD 21 int 21h

459

460 05AB 2E: 8E 06 002C mov ES, CS:2Ch ; загрузим в ES адрес окружения

461 05B0 B4 49 mov AH, 49h ; выгрузим из памяти окружение

462 05B2 CD 21 int 21h

463 05B4 72 0B jc \_notRemove

464

465 05B6 0E push CS

466 05B7 07 pop ES ; в ES - адрес резидентной программы

467 05B8 B4 49 mov AH, 49h ; выгрузим из памяти резидент

468 05BA CD 21 int 21h

469

470 05BC 72 03 jc \_notRemove

471 05BE EB 15 90 jmp \_unloaded

472

473 05C1 \_notRemove: ; не удалось выполнить выгрузку => вывод ошибки

474 05C1 B4 03 mov AH, 03h ; получаем позицию курсора

475 05C3 CD 10 int 10h

476 05C5 BD 0419r lea BP, noRemoveMsg

477 05C8 B9 001D mov CX, noRemoveMsg\_length

478 05CB B3 07 mov BL, 0111b

479 05CD B8 1301 mov AX, 1301h

480 05D0 CD 10 int 10h

481 05D2 EB 12 90 jmp \_2Fh\_exit

482

483 05D5 \_unloaded: ; выгрузка прошла успешно => вывод сообщения

484 05D5 B4 03 mov AH, 03h ; получаем позицию курсора

485 05D7 CD 10 int 10h

486 05D9 BD 03FDr lea BP, removedMsg

487 05DC B9 001C mov CX, removedMsg\_length

488 05DF B3 07 mov BL, 0111b

489 05E1 B8 1301 mov AX, 1301h

490 05E4 CD 10 int 10h

491

492 05E6 \_2Fh\_exit:

493 05E6 5B 5A 07 1F pop BX DX ES DS

494 05EA CF iret

495 05EB new\_int2Fh endp

496

497 ; обработчик прерывания int 1Ch

498 ; вызывается каждые 55 мс

499 05EB new\_int1Ch proc far

500 05EB 50 push AX

501 05EC 0E push CS

502 05ED 1F pop DS

503

504 05EE 9C pushf

505 05EF 2E: FF 1E 03ADr call dword ptr CS:[old\_int1ChOffset] ; вызываем стандартный обработчик прерывания

506

507 05F4 80 3E 0118r FF cmp signaturePrintingEnabled, true ; если нажата управляющая клавиша (в данном +

508 случае F1)

509 05F9 75 1B jne \_notToPrint

510

511 05FB 83 3E 0119r 25 cmp counter, printDelay\*1000/55 + 1 ; если кол-во "тактов" равно printDelay секундам

512 0600 74 03 je \_letsPrint

513

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 10

krbuff.asm

514 0602 EB 0E 90 jmp \_dontPrint

515

516 0605 \_letsPrint:

517 0605 F6 16 0118r not signaturePrintingEnabled

518 0609 C7 06 0119r 0000 mov counter, 0

519 060F E8 0016 call printSignature ; выводим подпись на экран

520

521 0612 \_dontPrint:

522 0612 FF 06 0119r inc counter ; увеличим значение счетчика на 1

523

524 0616 \_notToPrint:

525 0616 58 pop AX

526 0617 CF iret

527 0618 new\_int1Ch endp

528

529 ; выводит одну строку подписи

530 0618 printSignatureLine proc

531 0618 52 push DX

532 0619 8B 0E 011Br mov CX, signatureLineLength

533 061D B3 07 mov BL, 0111b ; цвет выводимого текста

534 061F B8 1301 mov AX, 1301h ; AH = 13h - номер ф-ии, AL = 01h - перемещение+

535 курсора

536 0622 CD 10 int 10h

537 0624 5A pop DX

538 0625 FE C6 inc DH

539 0627 C3 ret

540 0628 printSignatureLine endp

541

542 ; процедура вывода подписи

543 0628 printSignature proc

544 0628 50 52 51 53 06 54 55+ push AX DX CX BX ES SP BP SI DI

545 56 57

546

547 0631 33 C0 xor AX, AX ; обнуляем значения регистров

548 0633 33 DB xor BX, BX

549 0635 33 D2 xor DX, DX

550

551 0637 B4 03 mov AH, 03h ; чтение текущей позиции курсора

552 0639 CD 10 int 10h

553 063B 52 push DX ; помещаем информацию о положении курсора в стек

554

555 063C BA 090F mov DX, 090Fh ; NB! вверху: 000Fh, посередине: 090Fh, внизу: +

556 130Fh

557

558 063F \_actualPrint:

559 063F B4 0F mov AH, 0Fh ; чтение текущего видеорежима. в BH - текущая +

560 страница

561 0641 CD 10 int 10h

562

563 0643 0E push CS

564 0644 07 pop ES ; указываем ES на CS

565

566 0645 BD 01B9r lea BP, tableTop

567 0648 E8 FFCD call printSignatureLine ; выводим верх таблицы

568 064B BD 011Dr lea BP, signatureLine1

569 064E E8 FFC7 call printSignatureLine ; выводим первую строку

570 0651 BD 0151r lea BP, signatureLine2

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 11

krbuff.asm

571 0654 E8 FFC1 call printSignatureLine ; выводим вторую строку

572 0657 BD 0185r lea BP, signatureLine3

573 065A E8 FFBB call printSignatureLine ; выводим третью строку

574 065D BD 01EDr lea BP, tableBottom

575 0660 E8 FFB5 call printSignatureLine ; выводим низ таблицы

576

577 0663 33 DB xor BX, BX

578 0665 5A pop DX ; восстанавливаем из стека прежнее положение +

579 курсора

580 0666 B4 02 mov AH, 02h ; меняем положение курсора на первоначальное

581 0668 CD 10 int 10h

582

583 066A 5F 5E 5D 5C 07 5B 59+ pop DI SI BP SP ES BX CX DX AX

584 5A 58

585 0673 C3 ret

586 0674 printSignature endp

587

588 ; Основная часть программы

589 ; 1) установка видеорежима

590 ; 2) проверка, запущен ли резидент

591 ; 3) установка вектора прерываний

592 0674 \_initTSR:

593 0674 B4 03 mov AH, 03h

594 0676 CD 10 int 10h

595 0678 52 push DX

596 0679 B4 00 mov AH, 00h ; установка видеорежима

597 067B B0 83 mov AL, 83h

598 067D CD 10 int 10h

599 067F 5A pop DX

600 0680 B4 02 mov AH, 02h

601 0682 CD 10 int 10h

602

603 0684 E8 0084 call commandParamsParser ; читаем аргументы командной строки

604 0687 80 3E 0386r 02 cmp commandLineResult, 2 ; если результат = 2, значит была выведена +

605 справка

606 068C 75 03 jne \_shouldContinue ; соответственно, никаких других действий +

607 делать не нужно

608 068E EB 79 90 jmp \_exit

609 0691 \_shouldContinue:

610 ; @@@ следующий блок нужно оставить, только если нужна выгрузка при перезапуске @@@

611 ; mov AH, 0FFh

612 ; mov AL, 0

613 ; int 2Fh

614 ; cmp AH, 'i' ; проверка того, загружена ли уже программа

615 ; je \_remove

616 ; @@@ конец блока @@@

617 ; @@@ следующий блок нужно оставить, только если нужна выгрузка при аргументе /u @@@

618 ; cmp commandLineResult, 1 ; проверяем результат работы процедуры

619 ; jne \_go\_on

620 ; mov AH, 0FFh

621 ; mov AL, 0

622 ; int 2Fh

623 ; cmp AH, 'i' ; проверка того, загружена ли уже программа

624 ; je \_remove

625 ;

626 ; mov AH, 09h

627 ; lea DX, notInstalledMsg ; не была загружена

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 12

krbuff.asm

628 ; int 21h

629 ; int 20h

630 ; \_go\_on:

631 ; @@@ конец блока @@@

632 ; @@@ нужно раскомментировать во всех случаях, кроме выгрузки при перезапуске @@@

633 0691 B4 FF mov AH, 0FFh ; проверка того, что резидент уже запущен

634 0693 B0 00 mov AL, 0

635 0695 CD 2F int 2Fh

636 0697 80 FC 69 cmp AH, 'i' ; если запущена, то AH = 'i' (см. процедуру +

637 new\_int2Fh)

638 069A 74 64 je \_alreadyInstalled

639 ; @@@ конец блока @@@

640

641 069C B8 3509 mov AX, 3509h ; получить в ES:BX прерывания 09h

642 069F CD 21 int 21h

643 06A1 2E: 89 1E 03A9r mov word ptr CS:old\_int9hOffset, BX ; обработчик прерывания 09h

644 06A6 2E: 8C 06 03ABr mov word ptr CS:old\_int9hSegment, ES

645 06AB B8 2509 mov AX, 2509h ; установим вектор на прерывание 09h

646 06AE BA 0436r mov DX, offset new\_int9h

647 06B1 CD 21 int 21h

648

649 06B3 B8 351C mov AX, 351Ch ; получить в ES:BX прерывания 1Ch

650 06B6 CD 21 int 21h

651 06B8 2E: 89 1E 03ADr mov word ptr CS:old\_int1ChOffset, BX ; обработчик прерывания 1Ch

652 06BD 2E: 8C 06 03AFr mov word ptr CS:old\_int1ChSegment, ES

653 06C2 B8 251C mov AX, 251Ch ; установим вектор на прерывание 1Ch

654 06C5 BA 05EBr mov DX, offset new\_int1Ch

655 06C8 CD 21 int 21h

656

657 06CA B8 352F mov AX, 352Fh ; получить в ES:BX прерывания 2Fh

658 06CD CD 21 int 21h

659 06CF 2E: 89 1E 03B1r mov word ptr CS:old\_int2FhOffset, BX ; обработчик прерывания 2Fh

660 06D4 2E: 8C 06 03B3r mov word ptr CS:old\_int2FhSegment, ES

661 06D9 B8 252F mov AX, 252Fh ; установим вектор на прерывание 2Fh

662 06DC BA 055Er mov DX, offset new\_int2Fh

663 06DF CD 21 int 21h

664

665 06E1 BB 03B5r lea BX, installedMsg ; выводим сообщение, что всё ОК

666 06E4 E8 0061 call printStr

667

668 06E7 BA 0674r mov DX, offset \_initTSR ; остаемся в памяти и выходим из основной части

669 06EA CD 27 int 27h

670

671 06EC \_remove: ; выгрузка из памяти

672 06EC 06 push ES

673 06ED A1 002C mov AX, DS:[2Ch] ; PSP

674 06F0 8E C0 mov ES, AX

675 06F2 B4 49 mov AH, 49h ; хватит памяти чтоб остаться резидентом?

676 06F4 CD 21 int 21h

677 06F6 07 pop ES

678

679 06F7 B4 FF mov AH, 0FFh

680 06F9 B0 01 mov AL, 1

681 06FB CD 2F int 2Fh

682 06FD EB 0A 90 jmp \_exit

683 0700 \_alreadyInstalled: ; резидент уже запущен

684 0700 BB 03C8r lea BX, alreadyInstalledMsg

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 13

krbuff.asm

685 0703 E8 0042 call printStr

686 0706 EB 01 90 jmp \_exit

687 0709 \_exit: ; выход

688 0709 CD 20 int 20h

689

690 ; парсер аргментов командной строки. выводит справку.

691 ; устанавливает флаг commandLineResult:

692 ; 0 = всё ОК; 1 = нужна выгрузка; 2 = была выведена справка, не нужно загружать резидент

693 070B commandParamsParser proc

694 070B 0E push CS

695 070C 07 pop ES

696

697 070D BE 0080 mov SI, 80h ; SI = смещение командной строки

698 0710 AC lodsb ; получим кол-во символов

699 0711 0A C0 or AL, AL ; если 0 символов введено,

700 0713 74 2D jz \_paramParsingEnd ; то все в порядке

701

702 0715 \_nextChar:

703 0715 46 inc SI ; теперь SI указывает на первый символ строки

704

705 0716 80 3C 00 cmp [SI], BYTE ptr 0

706 0719 74 27 je \_paramParsingEnd

707

708 071B AD lodsw ; получаем два символа

709 071C 3D 3F2F cmp AX, '?/'

710 071F 74 16 je \_displayHelp

711 ; @@@ следует раскомментировать, если нужно выгружать по аргументу /u @@@

712 0721 3D 752F cmp AX, 'u/'

713 0724 74 07 je \_finishTSR

714 0726 3D 552F cmp AX, 'U/'

715 0729 74 02 je \_finishTSR

716 072B EB E8 jmp \_nextChar

717

718 072D \_finishTSR:

719 072D C6 06 0386r 01 mov commandLineResult, 1 ; флаг того, что необходимо выгрузить резидент

720 0732 EB E1 jmp \_nextChar

721 ; @@@ конец блока @@@

722

723 0734 EB 0C 90 jmp \_paramParsingEnd

724 0737 \_displayHelp:

725 0737 BB 0221r lea BX, helpMsg ; выводим справку

726 073A E8 000B call printStr

727 073D C6 06 0386r 02 mov commandLineResult, 2 ; флаг того, что резидент загружать не надо

728

729 0742 \_paramParsingEnd:

730 0742 C3 ret

731 0743 commandParamsParser endp

732

733 ; отображает символ из AL

734 0743 printChar proc

735 0743 B4 0E mov AH, 0EH

736 0745 CD 10 int 010H

737 0747 C3 ret

738 0748 printChar endp

739

740 ; отображает нуль-терминированную строку из [BX]

741 0748 printStr proc

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 14

krbuff.asm

742 0748 52 50 push DX AX

743 074A 8B 07 mov AX, [BX]

744 074C \_printStrLoop:

745 074C 3C 00 cmp AL, 0

746 074E 74 08 je \_printStrEnd

747 0750 E8 FFF0 call printChar

748 0753 43 inc BX

749 0754 8B 07 mov AX, [BX]

750 0756 EB F4 jmp \_printStrLoop

751 0758 \_printStrEnd:

752 0758 58 5A pop AX DX

753 075A C3 ret

754 075B printStr endp

755

756 075B code ends

757 end \_**start**

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 15

Symbol Table

Symbol Name Type Value

??DATE Text "04/18/22"

??FILENAME Text "krbuff "

??TIME Text "12:07:50"

??VERSION Number 030A

@CPU Text 0101H

@CURSEG Text CODE

@FILENAME Text KRBUFF

@WORDSIZE Text 2

ALREADYINSTALLEDMSG Byte CODE:03C8

CHANGEFONT Near CODE:0540

CHARTOCURSIVEINDEX Byte CODE:0398

COMMANDLINERESULT Byte CODE:0386

COMMANDPARAMSPARSER Near CODE:070B

COUNTER Word CODE:0119

CURSIVEENABLED Byte CODE:0387

CURSIVESYMBOL Byte CODE:0388

HELPMSG Byte CODE:0221

HELPMSGLENGTH Number 0165

IGNOREDCHARS Byte CODE:0103

IGNOREDLENGTH Byte CODE:0108

IGNOREENABLED Byte CODE:0109

INSTALLEDMSG Byte CODE:03B5

NEW\_INT1CH Far CODE:05EB

NEW\_INT2FH Near CODE:055E

NEW\_INT9H Far CODE:0436

NOREMOVEMSG Byte CODE:0419

NOREMOVEMSG\_LENGTH Number 001D

NOTINSTALLEDMSG Byte CODE:03E3

OLD\_INT1CHOFFSET Word CODE:03AD

OLD\_INT1CHSEGMENT Word CODE:03AF

OLD\_INT2FHOFFSET Word CODE:03B1

OLD\_INT2FHSEGMENT Word CODE:03B3

OLD\_INT9HOFFSET Word CODE:03A9

OLD\_INT9HSEGMENT Word CODE:03AB

PRINTCHAR Near CODE:0743

PRINTDELAY Number 0002

PRINTSIGNATURE Near CODE:0628

PRINTSIGNATURELINE Near CODE:0618

PRINTSTR Near CODE:0748

REMOVEDMSG Byte CODE:03FD

REMOVEDMSG\_LENGTH Number 001C

SAVEDSYMBOL Byte CODE:0399

SAVEFONT Near CODE:054F

SIGNATURELINE1 Byte CODE:011D

SIGNATURELINE2 Byte CODE:0151

SIGNATURELINE3 Byte CODE:0185

SIGNATURELINELENGTH Word CODE:011B

SIGNATUREPRINTINGENABLED Byte CODE:0118

TABLEBOTTOM Byte CODE:01ED

TABLETOP Byte CODE:01B9

TOGGLECURSIVE Near CODE:04F3

TRANSLATEENABLED Byte CODE:0117

TRANSLATEFROM Byte CODE:010A

TRANSLATELENGTH Byte CODE:0116

Turbo Assembler Version 3.1 04/18/22 12:07:50 Page 16

Symbol Table

TRANSLATETO Byte CODE:0110

TRUE Number 00FF

\_2FH\_DEFAULT Near CODE:056E

\_2FH\_EXIT Near CODE:05E6

\_ACTUALPRINT Near CODE:063F

\_ALREADYINSTALLED Near CODE:0700

\_ALREADYINSTALLED2FH Near CODE:0573

\_BLOCK Near CODE:04BF

\_CHECKIGNORED Near CODE:04B3

\_CHECKTRANSLATE Near CODE:04C7

\_CHECKTRANSLATELOOP Near CODE:04D5

\_DISPLAYHELP Near CODE:0737

\_DONTPRINT Near CODE:0612

\_EXIT Near CODE:0709

\_EXITTOGGLECURSIVE Near CODE:053D

\_F1 Near CODE:045E

\_F2 Near CODE:046F

\_F3 Near CODE:0483

\_F4 Near CODE:0494

\_FINISHTSR Near CODE:072D

\_GO Near CODE:045B

\_INITTSR Near CODE:0674

\_LETSPRINT Near CODE:0605

\_NEXTCHAR Near CODE:0715

\_NOTREMOVE Near CODE:05C1

\_NOTTOPRINT Near CODE:0616

\_PARAMPARSINGEND Near CODE:0742

\_PRINTSTREND Near CODE:0758

\_PRINTSTRLOOP Near CODE:074C

\_QUIT Near CODE:04EA

\_REMOVE Near CODE:06EC

\_RESTORESYMBOL Near CODE:052E

\_SHIFTTABLE Near CODE:0505

\_SHOULDCONTINUE Near CODE:0691

\_START Near CODE:0100

\_TEST\_FX Near CODE:045E

\_TRANSLATE Near CODE:04E1

\_TRANSLATEORIGNORE Near CODE:04A5

\_UNINSTALL Near CODE:0576

\_UNLOADED Near CODE:05D5

Groups & Segments Bit Size Align Combine Class

CODE 16 075B Para none CODE

# Исходный код Программы (UNLOADER)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; unloader.asm

;

; Сборка:

; tasm.exe /l unloader.asm

; tlink /t /x unloader.obj

;

; **Программа для выгрузки TSR из памяти,**

**Если по варианту группы выгрузка выполняется отдельным модулем.**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

code segment 'code'

assume CS:code, DS:code

org 100h

\_**start**:

mov AH, 0FFh

mov AL, 1

int 2Fh ; наше прерывание

int 20h ; выходим

code ends

end \_**start**

# Листинг Программы (UNLOADER)

Turbo Assembler Version 3.1 04/25/18 13:00:47 Page 1

unload.asm

1 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

2 ; unload.asm - выгружает TSR из памяти

3 ; **Если по варианту группы выгрузка выполняется ;отдельным модулем.**

4 ; tasm.exe /l unload.asm

5 ; tlink /t /x unload.obj

6 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

7

8 0000 code segment 'code'

9 assume CS:code, DS:code

10 org 100h

11 0100  **\_start**:

12

13 0100 B4 FF mov AH, 0FFh

14 0102 B0 01 mov AL, 1

15 0104 CD 2F int 2Fh

16 0106 CD 20 int 20h

17

18 0108 code ends

19 end **\_start**

Turbo Assembler Version 3.1 04/25/18 13:00:47 Page 2

Symbol Table

Symbol Name Type Value

??DATE Text "04/25/18"

??FILENAME Text "unload "

??TIME Text "13:00:47"

??VERSION Number 030A

@CPU Text 0101H

@CURSEG Text CODE

@FILENAME Text UNLOAD

@WORDSIZE Text 2

\_START Near CODE:0100

Groups & Segments Bit Size Align Combine Class

CODE 16 0108 Para none CODE