МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Операционные системы»

Тема: Исследование интерфейсов программных модулей

Студент гр. 7383 _____ Медведев И.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2019

Цель работы

Исследование интерфейса управляющей программы и загрузочных модулей. Этот интерфейс состоит в передаче запускаемой программе управляющего блока, содержащего адреса и системные данные. Так загрузчик строит префикс сегмента программы (PSP) и помещает его адрес в сегментный регистр. Исследование префикса сегмента программы (PSP) и среды, передаваемой программе.

Ход работы

В ходе работы были написаны некоторые функции и структуры данных, которые описаны в табл. 1-2.

Таблица 1 – Используемые функции

Название функции	Выполняемая задача
TETR_TO_HEX	Перевод числа из 2-ой в 16-ричную систему счисления.
BYTE_TO_HEX	Осуществляет перевод байта, помещенного в AL, в два символа в шестнадцатеричной системе счисления, помещая результат в AX.
WRD_TO_HEX	Переводит числа размером 2 байта в 16-ричную систему счисления. В AX – число, в DI – адрес последнего символа.
BYTE_TO_DEC	Переводит в 10-ную систему счисления, SI – адрес поля младшей цифры.
WRITE	Вывод сообщения на экран.

Окончание таблицы 1

GET_MEM	Получение и вывод строки с адресом недоступной памяти
GET_ENV	Получение и вывод строки с сегментным адресом среды, передаваемого программе
GET_TAIL	Получение и вывод строки с хвостом командной строки
GET_ENV_CONTENT	Получение и вывод строки с содержимым области среды
GET_PATH	Получение и вывод пути загружаемого модуля

Таблица 2 – Структуры данных

Название	Тип	Назначение
MEM	db	Строка «Memory address: h$»
ENV_ADRESS		Строка «Environment address: h$»
TAIL		Строка «Tail: $»
ENV		Строка «Environment contains: »
ENDL		Строка «$»
PATH		Строка «Path: $»

В ходе работы была написана программа, которая выводит на экран сегментный адрес первого байта недоступной памяти, сегментный адрес среды, передаваемой программе, хвост командной строки, содержимое

области среды в символьном виде и путь загружаемого модуля. Результаты работы представлены на рис. 1-2.

Рисунок 1 – Запуск .COM файла без хвоста.

Рисунок 2 – Запуск .COM файла с хвостом.

Вывод

В ходе выполнения лабораторной работы были исследованы: интерфейс управляющей программы и загрузочных модулей, а также префикс сегмента программы (PSP) и среды, передаваемой программе.

Ответы на контрольные вопросы

Сегментный адрес недоступной памяти.

1) На какую область памяти указывает адрес недоступной памяти?

На границу оперативной памяти и области, доступной для загрузки программ.

2) Где расположен этот адрес по отношению области памяти, отведенной программе?

Адрес указывает на байт, следующий за последним байтом выделенной памяти. Т.е. сразу после памяти, выделенной программе.

3) Можно ли в эту область памяти писать?

Да, т.к. в DOS нет защиты памяти.

Среда, передаваемая программе

1) Что такое среда?

Среда представляет собой последовательность строк имя – параметр, 00h. Имя и параметр являются последовательностью символов. Нулевой байт (00h) является концом данной строки.

2) Когда создается среда? Перед запуском приложения или в другое время?

Среда создается при загрузке DOS.

3) Откуда берется информация, записываемая в среду?

При работе с MS DOS, информация берется из файла autoexec.bat.

ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ

Lab2.asm

```
TESTPC     SEGMENT
           ASSUME  CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
               org 100h
START:     JMP     BEGIN

MEM  db 13, 10, "Memory address:     h$"
ENV_ADRESS  db 13, 10,  "Environment address:     h$"
TAIL db 13, 10, "Tail: $"
ENV db 13, 10, "Enviroment contains: ", 13, 10, "$"
ENDL db 13, 10, "$"
PATH db 13, 10, "Path: $"

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_TO_HEX       PROC near


            push  cx
            mov   al,ah
            call  TETR_TO_HEX
            xchg  al,ah
            mov   cl,4
            shr   al,cl
            call  TETR_TO_HEX
            pop   cx
            ret
BYTE_TO_HEX       ENDP
WRD_TO_HEX        PROC  near


            push  bx
            mov   bh,ah
            call  BYTE_TO_HEX
            mov   [di],ah
            dec   di
            mov   [di],al
            dec   di
            mov   al,bh
            xor   ah,ah
            call  BYTE_TO_HEX
            mov   [di],ah
            dec   di
            mov   [di],al
            pop   bx
            ret
WRD_TO_HEX        ENDP
```

```
BYTE_TO_DEC       PROC  near
            push  cx
            push  dx
            push  ax
            xor   ah,ah
            xor   dx,dx
            mov   cx,10
loop_bd:          div   cx
            or    dl,30h
            mov   [si],dl
            dec   si
            xor   dx,dx
            cmp   ax,10
            jae   loop_bd
            cmp   ax,00h
            jbe   end_l
            or    al,30h
            mov   [si],al
end_l:            pop   ax
            pop   dx
            pop   cx
            ret
BYTE_TO_DEC       ENDP


WRITE       PROC near
            mov   ah,09h
            int   21h
            ret
WRITE       ENDP


GET_MEM PROC near
      mov   ax, ds:[02h]
      lea di, MEM
      add di, 21
      call WRD_TO_HEX
      lea dx, MEM
      call WRITE
      ret
GET_MEM ENDP

GET_ENV PROC near
      mov ax, ds:[2Ch]
      lea di, ENV_ADRESS
      add di, 26
      call WRD_TO_HEX
      lea dx, ENV_ADRESS
      call WRITE
      ret
GET_ENV ENDP

GET_TAIL PROC near
      mov ax, ds:[80h]
      cmp al, 0
```

```
      je end_proc_tail
      lea dx, TAIL
      call WRITE
      mov cl, al
      mov di, 00h
A:    mov al, ds:[81h + di]
      mov dl, al
      mov ah, 02h
      int 21h
      inc di
      loop A
      jmp end_proc_tail

end_proc_tail:
      ret
GET_TAIL ENDP

GET_ENV_CONTENT PROC near
      lea dx, ENV
      call WRITE
      mov di, 00h
      mov bx, 2Ch
      mov ds, [bx]
read:
      cmp byte ptr [di], 00h
      jz add_endl
      mov dl, [di]
      mov ah, 02h
      int 21h
      jmp end_cont
add_endl:
      push ds
      mov cx, cs
      mov ds, cx
      lea dx, ENDL
      call WRITE
      pop ds
end_cont:
      inc di
      cmp byte ptr [di], 0001h
      jz end_get_env
      jmp read
end_get_env:
      ret
GET_ENV_CONTENT ENDP

GET_PATH PROC near
      push ds
      mov ax, cs
      mov ds, ax
      lea dx, PATH
      call WRITE
      pop ds
      add di, 2
loop_:
```

```
      cmp byte ptr  [di], 00h
      jz end_get_path
      mov dl, [di]
      mov ah, 02h
      int 21h
      inc di
      jmp loop_
end_get_path:
      ret
GET_PATH ENDP



BEGIN:
      call GET_MEM
      call GET_ENV
      call GET_TAIL
      call GET_ENV_CONTENT
      call GET_PATH

      xor al,al
      mov ah,4ch
      int   21h

TESTPC      ENDS
            END   START
```