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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Операционные системы»
Тема: Исследование интерфейсов командных модулей

Студент гр. 8381 _____ Облизанов А.Д.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

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.

Основные теоретические положения.

При начальной загрузке программы формируется PSP, который размещается в начале первого сегмента программы. PSP занимает 256 байт и располагается с адреса, кратного границе сегмента. При загрузке модулей типа COM все сегментные регистры указывают на адрес PSP. При загрузке модуля типа EXE сегментные регистры DS и ES указывают на PSP.

Формат PSP представлен в табл. 1.

Таблица 1 – Формат PSP

Смещение (16-ричн)	Длина поля (байт)	Содержимое поля
0	2	INT 20h
2	2	Сегментный адрес первого байта недоступной памяти. Программа не должна модифицировать содержимое памяти за этим адресом.
4	6	Зарезервировано
0A	4	Вектор прерывания 22h (IP, CS)
0E	4	Вектор прерывания 23h (IP, CS)
12	4	Вектор прерывания 24h (IP, CS)

Окончание табл. 1.

2C	2	Сегментный адрес среды, передаваемой программе.
5C		Область форматируется как стандартный неоткрытый блок управления файлом (FCB)
6C		Область форматируется как стандартный неоткрытый блок управления файлом (FCB). Перекрывается, если FCB с адреса 5Ch открыт.
80	1	Число символов в хвосте командной строки.
81		Хвост командной строки - последовательность символов после имени вызываемого модуля.

Выполнение работы.

Написание работы производилось на базе операционной системы Windows 10 в редакторе Visual Code. Сборка, отладка производились на базе операционной системы Windows XP через виртуальную машину.

Был написан текст исходного .COM модуля с именем, который обеспечивает вывод на экран следующей информации:

- Сегментный адрес недоступной памяти, взятый из PSP, в шестнадцатеричном виде
- Сегментный адрес среды, передаваемой программе, в шестнадцатеричном виде
- Хвост командной строки, в символьном виде
- Содержимое области среды в символьном виде
- Путь загружаемого модуля

Программа была отлажена и запущена на виртуальной машине с операционной системой Windows XP. Вывод программы представлен на рис. 1.

Рисунок 1 – Вывод .COM модуля

Исходный код модуля представлен в приложении А.

Контрольные вопросы

Сегментный адрес недоступной памяти

1. **На какую область памяти указывает адрес недоступной памяти?**

С адреса 0000h до 9FFFh адресное пространство отводится под пользовательские программы. Область недоступной памяти, которая начинается с адреса 9FFFh и заканчивается адресом FFFFh, является служебной – в ней DOS не может выделить память для пользовательской программы.

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

Из предыдущего пункта легко увидеть, что адрес недоступной памяти указывает на последний параграф памяти, отведенной для программ. Соответственно, недоступная память располагается сразу нее.

3. **Можно ли в эту область памяти писать?**

Так как операционная система DOS использует «реальный» режим процессора, в котором любому процессу доступна вся память, то можно, DOS не контролирует обращение программы к памяти.

Среда, передаваемая программе

1. **Что такое среда?**

Среда – область памяти, в которой в виде символьных строк хранятся значения переменных среды в формате «параметр=значение», 0, передаваемая программе при ее запуске.

Например, в первой строке указывается имя COMSPEC, которая определяет используемый командный процессор и путь к COMMAND.COM. Следующие строки содержат информацию, задаваемую командами PATH, PROMPT, SET.

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

Интерпретатор команд COMMAND.COM имеет свою среду, которую называют корневой средой. Она создается при запуске DOS. Передаваемая программе при запуске среда является копией среды родительского процесса. Поэтому, если COMMAND.COM запускает программу, то ей передается копия корневой среды.

3. **Откуда берется информация, записываемая в среду?**

Файл AUTOEXEX.BAT - системный пакетный файл, который содержит информацию о ключевых переменных среды (напр. команды PATH, PROMPT, SET). В MS-DOS AUTOEXEC.BAT исполняется во время загрузки операционной системы. По информации из файла создается корневая среда. При запуске программы ей передается копия корневой среды.

Выводы.

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

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. LR2.ASM

```
AO SEGMENT ; Hello, I am Alexandr O.
      ASSUME  CS:AO, DS:AO, ES:NOTHING, SS:NOTHING
      ORG 100H

START: JMP BEGIN

INACCESSIBLE_MEMORY_INFO     db    "Inaccessible memory adress:       ", 13, 10,
"$"
ENVIRONMENT_INFO             db    "Program environment adress:       ", 13, 10,
"$"
LINE_TAIL_INFO                     db    "Command line tail:                 ",
13, 10, "$"
ENVIRONMENT_CONTENT_INFO     db  "Program environment content:", 13, 10, "$"
ENVIRONMENT_CONTENT_END      db   "Program environment content ended", 13, 10,
"$"
PATH_INFO                          db  "Path:", 13, 10, "$"
NO_TAIL_INFO                       db    "No command line tail", 13, 10, "$"
TAIL_INFO                          db  " $"
CONTENT_NEW_LINE             db    13, 10, "$"

; There is some basic procedures...

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_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
            push  ax
            call  BYTE_TO_HEX
            mov         [di], ah
            dec         di
            mov         [di], al
            dec         di
            pop         ax
            mov         ah, al
            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

BEGIN:
            push  AX
            push  DX

INACCESSIBLE_MEMORY_PRINT:
            mov   DI, offset INACCESSIBLE_MEMORY_INFO
            add   DI, 32
            mov   AH, DS:[02h]
            mov         AL, DS:[03h]
            call  WRD_TO_HEX
            mov   DX, offset INACCESSIBLE_MEMORY_INFO
            call  PRINT_STRING

ENVIRONMENT_PRINT:
            mov   DI, offset ENVIRONMENT_INFO
            add   DI, 32
            mov   AH, DS:[2Ch]
            mov         AL, DS:[2Dh]
            call  WRD_TO_HEX
            mov   DX, offset ENVIRONMENT_INFO
            call  PRINT_STRING

LINE_TAIL_PRINT:
            mov   DX, offset LINE_TAIL_INFO
            call  PRINT_STRING
            mov   AL, DS:[80h]
            cmp         AL, 0
            je          NO_TAIL
            mov   DX, offset TAIL_INFO
            mov   DI, offset TAIL_INFO
            mov         SI, 0
TAIL_CYCLE:
            mov         AL, DS:[81h + SI]
            mov   AH, 02h
            int   21h
            inc         SI
            cmp   SI, AX
            jne         TAIL_CYCLE
NO_TAIL:
            mov   DX, offset NO_TAIL_INFO
            call  PRINT_STRING
```

```
ENVIRONMENT_CONTENT_PRINT:
            mov         DX, offset ENVIRONMENT_CONTENT_INFO
            call  PRINT_STRING
            mov   BX, 2Ch
            mov   ES, [BX]
            xor   SI, SI
            xor   AX, AX
            mov         DX, offset CONTENT_NEW_LINE
LINE_PRINT:
            mov   AL, ES:[SI]
            cmp   AL, 0
            jne         LINE_SYMB_PRINT
            mov   DX, offset CONTENT_NEW_LINE
            call  PRINT_STRING
LINE_SYMB_PRINT:
            mov   DL, AL
            xor   AX, AX
            mov   AH, 02h
            int   21h
            inc   SI
            mov   AX, ES:[SI]
            cmp         AX, 0001h;
            je          LINE_END
            jmp   LINE_PRINT
LINE_END:
            mov   DX, offset ENVIRONMENT_CONTENT_END
            call  PRINT_STRING

PATH_PRINT:
            mov   DX, offset PATH_INFO
            call  PRINT_STRING
            add   SI, 2
PATH_SYMB_PRINT:
            mov   AL, ES:[SI]
            cmp         AL, 0
            je          ENDING
            mov         DL, AL
            mov         AH, 02h
            int         21h
            inc         SI
            jmp         PATH_SYMB_PRINT

ENDING:
            pop   DX
            pop         AX
            xor   AL, AL
```

```
            mov   AH, 4Ch
            int   21h
AO ENDS
END START
```