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. 8383 _____ Костарев К.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование интерфейса управляющей программы и загрузочных модулей, а также префикса сегмента программы и среды, передаваемой программе.

Основные теоретические сведения.

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. Формат PSP представлен в табл. 1.

Таблица 1 – Формат PSP

Смещение	Длина поля (байт)	Содержимое поля
0	2	int 20h
2	2	Сегментный адрес первого байта недоступной памяти
4	6	Зарезервировано
0Ah	4	Вектор прерывания 22h
0Eh	4	Вектор прерывания 23h
12h	4	Вектор прерывания 24h
2Ch	2	Сегментный адрес среды, передаваемой программе
5Ch		Форматируется как стандартный неоткрытый блок управления файлом (FCB)
6Ch		Перекрывается, если FCB с адреса 5Ch открыт
80h	1	Число символов в хвосте командной строки
81h		Хвост командной строки

Выполнение работы.

Для выполнения лабораторной работы был написан и отлажен программный модуль типа .COM, который выбирает и печатает на экран следующую информацию:

1) Сегментный адрес недоступной памяти в шестнадцатеричном виде;
2) Сегментный адрес среды, передаваемой программе, в шестнадцатеричном виде;
3) Хвост командной строки в символьном виде;

4) Содержимое области среды в символьном виде;
5) Путь загружаемого модуля.

Результат работы программы представлен на рис. 1, исходный код программы в Приложении А.

Рисунок 1 – Результат работы программы

Сегментный адрес недоступной памяти:

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

   Адрес недоступной памяти указывает на служебную часть памяти, эту память DOS не может выделить под программу. Сам адрес указывает на сегментный адрес последнего параграфа памяти, используемого DOS для запуска программ.

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

   Расположен сразу за областью памяти, которую DOS отводит пользовательским программам.

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

   Возможно, так как DOS не контролирует обращение программ к памяти.

Среда, передаваемая программе:

1) *Что такое среда?*

   Это последовательность символьных строк вида [имя] = [параметр]. COMSPEC определяет путь к COMMAND.COM, PATH определяет пути к программным файлам, которые будут вызваны. Признаком конца строки является байт нулей.

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

При запуске DOS создается корневая среда, относящаяся к COMMAND.COM. Затем, когда COMMAND.COM запускает пользовательскую программу или одна программа запускает другую создается процесс, который получает собственный экземпляр блока среды, по умолчанию являющийся копией среды родителя.

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

   Из среды родителя.

Выводы.

В ходе выполнения данной лабораторной работы был исследован интерфейс управляющей программы и загрузочных модулей, а также префикс сегмента программы и среды, передаваемой программе. Для этого был написан программный модуль типа .COM, который непосредственно обращается к некоторым сегментам памяти в PSP и выводит на экран информацию из них.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД РЕАЛИЗОВАННОЙ ПРОГРАММЫ

```
LR2 SEGMENT
ASSUME CS:LR2, DS:LR2, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
ENTER db 13, 10, '$'
LOCKMEMORY db 'Locked memory addres: $'
ENVIROMENT db 'Enviroment addres: $'
TAIL db 'Command line tail: $'
NO_TAIL db 'is not command line tail $'
ENVIROMENT_CONTENT db 'Enviroment content: $'
PATH db '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 AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX
    pop CX
    ret
BYTE_TO_HEX ENDP

WRITE_SYMBOL PROC near
    push AX
    mov AH, 02H
    int 21H
    pop AX
    ret
WRITE_SYMBOL ENDP

WRITE_STRING PROC near
    push AX
    mov AH, 09H
    int 21H
    pop AX
    ret
```

```
WRITE_STRING ENDP

WRITE_HEX PROC near
    push AX
    mov AL, AH
    call BYTE_TO_HEX
    mov DL, AH
    call WRITE_SYMBOL
    mov DL, AL
    call WRITE_SYMBOL
    pop AX
    call BYTE_TO_HEX
    mov DL, AH
    call WRITE_SYMBOL
    mov DL, AL
    call WRITE_SYMBOL
    ret
WRITE_HEX ENDP

WRITE_LOCKMEMORY PROC near
    push AX
    push DX
    mov DX, offset LOCKMEMORY
    call WRITE_STRING
    mov AX, DS:[02H]
    call WRITE_HEX
    mov DX, offset ENTER
    call WRITE_STRING
    pop DX
    pop AX
    ret
WRITE_LOCKMEMORY ENDP

WRITE_ENVIROMENT PROC near
    push AX
    push DX
    mov DX, offset ENVIROMENT
    call WRITE_STRING
    mov AX, DS:[2CH]
    call WRITE_HEX
    mov DX, offset ENTER
    call WRITE_STRING
    pop DX
    pop AX
    ret
WRITE_ENVIROMENT ENDP

WRITE_TAIL PROC near
    push AX
    push SI
```

```
    push CX
    push DX
    mov DX, offset TAIL
    call WRITE_STRING
    xor CX, CX
    mov CL, DS:[80H]
    cmp CL, 0
    jne RETAIL
    mov DX, offset NO_TAIL
    call WRITE_STRING
    jmp POPING_TAIL
RETAIL:
    xor SI, SI
    xor AX, AX
WRITING_TAIL:
    mov AL, DS:[81H + SI]
     call WRITE_SYMBOL
     inc SI
     loop WRITING_TAIL
POPING_TAIL:
    mov DX, offset ENTER
    call WRITE_STRING
    pop DX
    pop CX
    pop SI
    pop AX
    ret
WRITE_TAIL ENDP

WRITE_ENVIROMENT_CONTENT_AND_PATH PROC near
    push AX
    push SI
    push DX
    push BX
    push ES
    mov DX, offset ENVIROMENT_CONTENT
    call WRITE_STRING
    xor SI, SI
     mov BX, 2CH
     mov ES, [BX]
WRITING_CONTENT:
     cmp BYTE PTR ES:[SI], 0H
     je NEXT_CONTENT
     mov AL, ES:[SI]
    mov DL, AL
     call WRITE_SYMBOL
     jmp CHECKING
NEXT_CONTENT:
     mov DX, offset ENTER
     call WRITE_STRING
```

```
CHECKING:
      inc SI
      cmp WORD PTR ES:[SI], 0001H
    je WRITE_PATH
      jmp WRITING_CONTENT
WRITE_PATH:
      mov DX, offset PATH
      call WRITE_STRING
      add SI, 2
WRITING_PATH:
      cmp BYTE PTR ES:[SI], 00H
      je POPING_CONTENT
      mov AL, ES:[SI]
    mov DL, AL
      call WRITE_SYMBOL
      inc SI
      jmp WRITING_PATH
POPING_CONTENT:
    pop ES
    pop BX
    pop DX
    pop SI
    pop AX
    ret
WRITE_ENVIROMENT_CONTENT_AND_PATH ENDP

BEGIN:
    call WRITE_LOCKMEMORY
    call WRITE_ENVIROMENT
    call WRITE_TAIL
    call WRITE_ENVIROMENT_CONTENT_AND_PATH
    xor AL,AL
    mov AH,4CH
    int 21H
LR2 ENDS
END START
```