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.

Студентка гр. 9383 _____ Лысова А.М.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

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.

Задание.

Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа **.COM,** который выбирает и распечатывает следующую информацию:

1. Сегментный адрес недоступной памяти, взятый из PSP, 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.

Сохраните результаты, полученные программой, и включите их в отчет.

Оформление отчета в соответствии с требованиями. В отчет включите скриншот с запуском программы и результатами.

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. Именно по этой причине значения этих регистров в модуле **.EXE** следует переопределять.

Формат PSP:

Смещение	**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.
0Ah (10)	4	Вектор прерывания 22h (IP, CS).
0Eh (14)	4	Вектор прерывания 23h (IP, CS).
12h (18)	4	Вектор прерывания 24h (IP, CS).
2Ch (44)	2	Сегментный адрес среды, передаваемой программе.
5Ch		Область форматируется как стандартный неоткрытый блок управления файлом (FCB).
6Ch		Область форматируется как стандартный неоткрытый блок управления файлом (FCB). Перекрывается, если FCB с адреса 5Ch открыт.
80h	1	Число символов в хвосте командной строки.
81h		Хвост командной строки — последовательность символов после имени вызываемого модуля.

Область среды содержит последовательность символьных строк вида:

имя = параметр

Каждая строка завершается байтом нулей.

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.

Среда заканчивается также байтом нулей. Таким образом, два нулевых байта являются признаком конца переменных среды. Затем идут два байта, содержащие 00h, 01h, после которых располагается маршрут загруженной программы. Маршрут также заканчивается байтом 00h.

Выполнение работы.

1. Был написан текст исходного загрузочного .COM модуля, выполняющего поставленную задачу.
2. Результаты были сохранены и проанализированы.
3. Был написан отчет в соответствии с требованиями.

Используемые функции.

- TETR_TO_HEX — шаблонная функция, которая переводит десятичное число в код символа.
- BYTE_TO_HEX — шаблонная функция, которая переводит байт в шестнадцатеричной СС в код символа.
- WRD_TO_HEX — шаблонная функция, которая переводит шестнадцатеричное число в символьный код.
- BYTE_TO_DEC — шаблонная функция, которая переводит байт в шестнадцатеричной СС в символьный код десятичной СС.
- PRINT_STRING — функция, которая печатает строку на экран.
- DATA_INAC_MEMORY — функция, которая печатает на экран информацию о недоступной памяти.
- DATA_ENV — функция, которая печатает на экран информацию о среде.
- DATA_TAIL — функция, которая печатает на экран хвост командной строки в символьном виде.

- DATA_CONTENT — функция, которая печатает на экран содержимое области среди и путь загружаемого модуля.

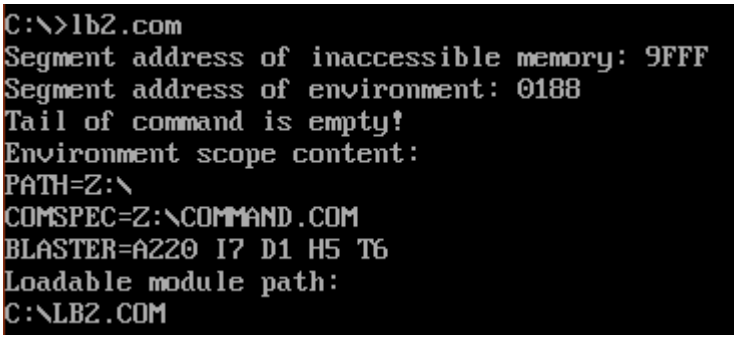

Рисунок 1 -- выполнение программы без аргументов.

Рисунок 2 -- выполнение программы с аргументом командной строки.

Контрольные вопросы.

Сегментный адрес недоступной памяти.

1) На какую область памяти указывает адрес недоступной памяти?

На первый байт, находящийся сразу после памяти, которая была отведена под программу.

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

В сторону больших адресов, то есть их увеличения.

3) Можно ли в эту область памяти писать?

Можно, потому что в DOS отсутствует контроль доступа к памяти, следовательно любая программа может перезаписать память, даже если она отведена не под нее.

Среда, передаваемая программе.

1) Что такое среда?

Область памяти, хранящая переменные среды в виде символьных строк. В этих переменных записана информация о состоянии системы, например, путь к домашней директории, путь к модулю COMMAND.COM, данные об используемом командном процессоре и др.

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

Среда создается при загрузке ОС, после чего при запуске конкретной программы она копируется в его адресное пространство и может быть изменена в соответствии с требованиями программы.

3) Откуда берется информация, записываемая в среду?

Из системного пакетного файла AUTOEXEC.BAT, который расположен в корневом каталоге загрузочного устройства.

Выводы.

Был исследован интерфейс управляющей программы и загрузочных модулей, а также префикс сегмента программы (PSP) и среда, передаваемая программе.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД.

Название файла: lb2.asm

```
TESTPC      SEGMENT
      ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
      ORG 100H
START:      JMP BEGIN

SEG_INAC_MEMORY db 'Segment address of inaccessible memory:          ', 0DH,
0AH, '$'
SEG_ENV db 'Segment address of environment:          ', 0DH, 0AH, '$'
TAIL_COM db 'Tail of command string:                                ', 0DH, 0AH,
'$'
ENV_SCOPE db 'Environment scope content: ', 0DH, 0AH, '$'
LOAD_PATH db 'Loadable module path: ', 0DH, 0AH, '$'
NULL_TAIL db 'Tail of command is empty! ', 0DH, 0AH, '$'
END_STRING db 0DH, 0AH, '$'

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

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
      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
      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 AL, 00h
      je end_1
      or AL, 30h
      mov [SI], AL
end_1:
      pop DX
      pop CX
      ret
BYTE_TO_DEC ENDP

PRINT_STRING      PROC near
      push AX
      mov ah, 09h
      int 21h
      pop AX
      ret
PRINT_STRING      ENDP

DATA_INAC_MEMORY  PROC near
      mov AX, DS:[02h]
      mov DI, offset SEG_INAC_MEMORY
      add DI, 43
      call WRD_TO_HEX
      mov DX, offset SEG_INAC_MEMORY
      call PRINT_STRING
      ret
DATA_INAC_MEMORY  ENDP

DATA_ENV    PROC near
      mov AX, DS:[2Ch]
      mov DI, offset SEG_ENV
      add DI, 35
      call WRD_TO_HEX
      mov DX, offset SEG_ENV
      call PRINT_STRING
      ret
DATA_ENV    ENDP

DATA_TAIL   PROC near
      xor CX, CX
      mov CL, DS:[80h]
      mov SI, offset TAIL_COM
      add SI, 25
      cmp CL, 0h
      je EMPTY
      xor DI, DI
      xor AX, AX
```

```
READ:
      mov AL, DS:[81h+DI]
      inc DI
      mov [SI], AL
      inc SI
      loop read
      mov DX, offset TAIL_COM
      jmp PRINT_TAIL
EMPTY:
      mov DX, offset NULL_TAIL
PRINT_TAIL:
      call PRINT_STRING
      ret
DATA_TAIL   ENDP

DATA_CONTENT      PROC near
mov DX, offset ENV_SCOPE
      call PRINT_STRING
      xor DI, DI
      mov DS, DS:[2Ch]

READ_STRING:
      cmp byte ptr [DI], 00h
      jz END_CONTENT
      mov DL, [DI]
      mov AH, 02h
      int 21h
      jmp FIND

END_CONTENT:
      cmp byte ptr [DI+1], 00h
      jz FIND
      push DS
      mov CX, CS
      mov DS, CX
      mov DX, offset END_STRING
      call PRINT_STRING
      pop DS

FIND:
      inc DI
      cmp word ptr [DI], 0001h
      jz PATH
      jmp READ_STRING

PATH:
      push DS
      mov AX, CS
      mov DS, AX
      mov DX, offset LOAD_PATH
      call PRINT_STRING
      pop DS
      add DI, 2
LOOP_PATH:
      cmp byte ptr [DI], 00h
      jz EXIT
      mov DL, [DI]
      mov AH, 02h
      int 21h
      inc DI
      jmp LOOP_PATH

EXIT:
```

```
      ret
DATA_CONTENT      ENDP

BEGIN:
      xor AX, AX

      call DATA_INAC_MEMORY
      call DATA_ENV
      call DATA_TAIL
      call DATA_CONTENT

      xor AL, AL
      mov AH, 4Ch
      int 21h

TESTPC      ENDS
      END START
```