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. 0382 _____ Андрющенко К.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

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.

Шаг 1. 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.

Шаг 2. Оформление отчета в соответствии с требованиями. В отчет включите скриншот с запуском программы и результатами.

Порядок выполнения работы.

1. Формируем .COM файл. Используем шаблон из лабораторной работы 1.
2. Для вывода сохраненных сообщение используем макрос PRINT_STRING.
3. Для вывода области памяти создаем макрос, заканчивающейся байтом нулей.
4. Выводим адрес недоступной памяти.
5. Выводим адрес среды.
6. Вывод хвоста командной строки в символьном виде.
7. Выводим содержимое области среды в символьном виде.

8. И путь загружаемого модуля.
9. Для безопасности выполнения процедур в них сохраняются рагистры в стек.

Тестирование.

Рис.1 Результаты тестирования

Вывод: программа работает корректно.

Контрольные вопросы.

Сегментный адрес недоступной памяти

1. На какую область памяти указывает адрес недоступной памяти?

   На сегмент памяти, находящийся за выделенной программе памяти.
2. Где расположен этот адрес по отношению области памяти, отведенной программе?

   После памяти, выделенной программе первый байт.
3. Можно ли в эту область памяти писать?

   Механизмы защиты памяти отсутствуют, поэтому, можно, но нежелательно.

Среда, передаваемая программе

1. Что такое среда?

   Специальная область памяти DOS, в которой хранятся наборы строк символов, использующиеся программами, формата имя-переменной (строка символов, не содержащая знаков равенства и пробелов) = значение (любая строка символов).
2. Когда создается среда? Перед запуском приложения или в другое время?

При запуске ОС. В случае запуска программы содержимое родительской среды копируется, имеется возможность дополнения среды родительской программой.

1. Откуда берется информация, записываемая в среду?

   Переменные окружения определяются командой SET (файл Autoexec.bat) при запуске ОС. Исключения:

   - CONFIG - определяется в файле Config.sys,

   - PROMPT - определяется отдельной командой DOS - оболочки,

   - PATH - задается отдельно в файле Autoexec.bat.

Вывод.

В результате работы был исследован префикс сегмента программы и среды, передаваемой программе и интерфейс управляющей программы и загрузочных модулей.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД МОДУЛЕЙ

Файл LAB2_C.ASM

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

MEM_ADDR_DATA db 'Segment address of unavailable memory: ',0DH,0AH,'$'
ENV_ADDR_DATA db 'Segment address of the environment: ',0DH,0AH,'$'
TAIL_DATA db 'Command line tail:','$'
EMPTY_TAIL_DATA db 'The tail of the command is empty',0DH,0AH,'$'
CONT_ENV_DATA db 'The contents of the environment area in symbolic form:',0DH,0AH,'$'
PATH_DATA db 'The path of the loaded module:','$'

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_l
    or AL,30h
    mov [SI],AL
end_l: pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP

PRINT_STRING PROC near
    mov AH, 09h
    int 21h
    ret
PRINT_STRING ENDP

PRINT_SYMB PROC near
    push ax
    mov ah, 02h
    int 21h
    pop ax
    ret
PRINT_SYMB ENDP

MEM_ADDR PROC near
    push AX
     push CX
     push DX
     push DI
     push ES

    mov ax, ds:[2h]
    mov di, offset MEM_ADDR_DATA + 42
    call WRD_TO_HEX
    mov dx, offset MEM_ADDR_DATA
    call PRINT_STRING

    pop ES
     pop DI
     pop DX
     pop CX
     pop AX
    ret
MEM_ADDR ENDP
```

```
ENV_ADDR PROC near
    push AX
     push CX
     push DX
     push DI
     push ES

    mov ax, ds:[2Ch]
    mov di, offset ENV_ADDR_DATA + 39
    call WRD_TO_HEX
    mov dx, offset ENV_ADDR_DATA
    call PRINT_STRING

    pop ES
     pop DI
     pop DX
     pop CX
     pop AX
    ret
ENV_ADDR ENDP

TAIL PROC near
    push AX
     push CX
     push DX
     push DI
     push ES

    mov cl, ds:[80h]
    cmp cl, 0h
    je empty_t
    mov dx,offset TAIL_DATA
    call PRINT_STRING
    mov di,81h
loop_t:
    mov dl, ds:[di]
    call PRINT_SYMB
    inc di
    loop loop_t
    mov dl, 0Dh
    call PRINT_SYMB
    mov dl, 0Ah
    call PRINT_SYMB
    jmp final_t
empty_t:
    mov dx, OFFSET EMPTY_TAIL_DATA
    call PRINT_STRING
final_t:

    pop ES
     pop DI
     pop DX
     pop CX
     pop AX
    ret
TAIL ENDP
```

```
CONT_ENV PROC near
    push AX
     push CX
     push DX
     push DI
     push ES

    mov dx, offset CONT_ENV_DATA
    call PRINT_STRING
    mov es, ds:[2Ch]
    xor di, di
loop_c:
    mov dl, es:[di]
    cmp dl, 0h
    je final_c
    call PRINT_SYMB
    inc di
    jmp loop_c
final_c:
    mov dl, 0Dh
    call PRINT_SYMB
    mov dl, 0Ah
    call PRINT_SYMB
    inc di
    mov dl, es:[di]
    cmp dl, 0h
    jne loop_c

    pop ES
     pop DI
     pop DX
     pop CX
     pop AX
    ret
CONT_ENV ENDP

PATH PROC near
    push AX
     push CX
     push DX
     push DI
     push ES

    mov di, 3
    mov dx, offset PATH_DATA
    call PRINT_STRING
loop_p:
    mov dl, es:[di]
    cmp dl, 0h
    je final_p
    call PRINT_SYMB
    inc di
    jmp loop_p
final_p:
    mov dl, 0Dh
    call PRINT_SYMB
```

```
    mov dl, 0Ah
    call PRINT_SYMB

    pop ES
     pop DI
     pop DX
     pop CX
     pop AX
    ret
PATH ENDP


; КОД
BEGIN:
    call MEM_ADDR
    call ENV_ADDR
    call TAIL
    call CONT_ENV
    call PATH
    xor AL,AL
    mov AH,4Ch
    int 21H
TESTPC ENDS
    END START
ere  Errors
```