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 _____ Самулевич В.А.

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.

Выполнение работы.

Для решения поставленной задачи были написаны 3 функции:

1) print_adresses

Эта функция печатает сегментный адрес недоступной памяти и среды.

Для этого она сначала извлекает соответствующие значения из PSP(DS:02 и DS:02ch соответственно),после чего записывает их в конец строк-сообщений(с помощью функции WRD_TO_HEX) и выводит изменённые строки на экран.

2) print_arguments

Выводит хвост командной строки в символьном виде.

Алгоритм: Из DS:80h в CX помещается длина хвоста, после чего с помощью команды REP movsb, происходит копирование символов из PSP (начиная с 81h), в строку buffer.После завершения копирования, buffer выводится на экран.

3) print_enviroment

Была создана для печати переменных среды, а также пути до загружаемого модуля.

Алгоритм:

I. В ES загружается адрес сегмента среды (Он находится в ячейке DS:02ch).

II. Пока не встретятся два нулевых байта, вызывается функция print_sentence, которая выводит на экран все символы до байта 00.Таким образом выведутся все переменные среды.

III. После завершения переменных среды функция print_sentence вызывается еще один раз. В этом случае она выведет путь до загружаемого модуля.

Тестирование.

Рис.1.Результат работы программы.

Контрольные вопросы.

1) Сегментный адрес недоступной памяти
   I. На какую область памяти указывает адрес недоступной памяти?
   На сегмент, расположенный сразу после выделенной программе памяти.

   II. Где расположен это адрес по отношению области памяти, отведённой программе?
   Первый байт после выделенной программе памяти.

   III. Можно ли в эту область памяти писать?
   Да, можно.

2) Среда, передаваемая программе
   I. Что такое среда?

Среда-область памяти, где хранятся определенные параметры системы, называемые переменными среды.

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

Среда создается при запуске ОС.В свою очередь, при запуске приложения, создается копия этой среды, в которую могут добавляться дополнительные параметры для данного приложения.

III. Откуда берется информация, записываемая в среду?

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

Выводы.

Были исследованы интерфейсы управляющей программы и загрузочных модулей. Также была разработана программа, выводящая на экран сегментный адрес недоступной памяти, сегментный адрес среды, хвост командной строки, содержимое области среды и путь загружаемого модуля.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab2.asm

```
CODE SEGMENT
ORG 100h
ASSUME CS:CODE,DS:CODE,SS:CODE,ES:NOTHING

start: jmp MAIN
    blocked_memory DB "Inaccessible memory adress:    h",0DH,0AH,'$'
    enviroment     DB "Enviroment segment adress:    h",0DH,0AH,'$'
    arguments_message DB "Command line arguments:$"
    content  DB "Enviroment content:",0DH,0AH,'$'
    path DB "Path of the loaded module:$"
    buffer DB 259 DUP(?)



MAIN:
    call print_adresses
    call print_arguments
    call print_enviroment
    xor AL,AL
    mov AH,4Ch
    int 21h



print_enviroment PROC NEAR
    push SI
    push DX
    push ES
    mov DX,offset content
    call print
    mov SI,0
    mov ES, DS:02ch
enviroment_cycle:
    cmp [ES:SI], BYTE PTR 0
    je enviroment_stop
    call print_sentence
    inc SI
    jmp enviroment_cycle

enviroment_stop:
    mov DX,offset path
    call print
    add SI,3
    call print_sentence
    pop ES
    pop DX
    pop SI
    ret
print_enviroment ENDP
```

```
print_sentence PROC NEAR
    push DI
    push DX
    push AX

    mov DI,offset buffer
sentence_cycle:
    cmp [ES:SI],BYTE PTR 0
    je sentence_stop
    mov AL,[ES:SI]
    mov [DI],AL
    inc SI
    inc DI
    jmp sentence_cycle
sentence_stop:
    mov [DI],  BYTE PTR 0Dh
    inc DI
    mov [DI],  BYTE PTR 0Ah
    inc DI
    mov [DI], BYTE PTR 36
    mov DX, offset buffer
    call print

    pop AX
    pop DX
    pop DI
    ret
print_sentence ENDP

print_adresses PROC NEAR
    push AX
    push DX
    push DI

    mov AX,DS:02h
    mov DI,offset blocked_memory
    add DI,30
    call WRD_TO_HEX
    mov DX,offset blocked_memory
    call print
    mov AX,DS:02cH
    mov DI,offset enviroment
    add DI,29
    call WRD_TO_HEX
    mov DX,offset enviroment
    call print
    pop DI
    pop DX
    pop AX
    ret
print_adresses ENDP


print_arguments PROC NEAR
    push DX
    push CX
    push SI
    push DI
```

```
    mov CX,0
    mov CL,DS:80h
    mov SI,81h
    mov DI,offset buffer
    REP movsb
    mov [DI],  BYTE PTR 0Dh
    inc DI
    mov [DI],  BYTE PTR 0Ah
    inc DI
    mov [DI], BYTE PTR 36

    mov DX,offset arguments_message
    call print

    mov DX,offset buffer
    call print

    pop DI
    pop SI
    pop CX
    pop DX
    ret
print_arguments ENDP


print PROC NEAR
    push AX
    mov AH,09h
    int 21h
    pop AX
    ret
print 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 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_BYTE PROC near
```

```
        call BYTE_TO_HEX
        mov [DI],AH
        dec DI
        mov [DI],AL
        ret
    WRITE_BYTE 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


    CODE ENDS
END start
```