**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 _____ Кривенцова Л.С.

Преподаватель _____ Губкин А.Ф.

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.**

Написать и отладить программный модуль типа .COM, который выбирает и распечатывает 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.

**Выполнение работы.**

Исходный код содержится в файле lb2.asm.

Регистр DS указывает на начало префикса сегмента программы, следовательно, получить информацию из PSP мы можем как DS:[смещение].

- Функция inaccessible_memory:

  Чтобы узнать адрес недоступной памяти записываем значение из DS:[2H] в AX. Затем с помощью функции WRD_TO_HEX получаем строковое представление шестнадцатеричного числа и выводим на экран.

- Функция address_of_environment:

  Функция работает аналогично. Только чтобы узнать сегментный адрес среды, передаваемой программе смещение не 2H, а 2CH.

- Функция command_line_tail:

По адресу DS:[80H] получаем длину хвоста командной строки и записываем в CL. Если это значение равно нулю, выводится соответствующее сообщение (offset Task3_2). В противном случае хвост считывается и выводится посимвольно.

- Функция environment_area:

Организован цикл считывания посимвольно, с помощью регистра AX, используемого как очередь (2 байта), чтобы отследить конец – 2 байта, равных 0. В итерации этого цикла выводится символ из AL, куда далее записывается символ из AH, а в AH помещается следующий символ.

Считывание пути загружаемого модуля происходит аналогично, после того как в очереди оказывается два нулевых байта (конец содержимого области среды).

Рис. 1,2 – Результаты работы программы.



**Контрольные вопросы.**

**Сегментный адрес недоступной памяти:**

1. На какую область памяти указывает адрес недоступной памяти?

Сегмент оперативной памяти, расположенный после памяти, отведенной программе.

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

Сегмент оперативной памяти расположен сразу после памяти, отведенной программе.

3. Можно ли в эту область памяти писать?

Да, т.к. защита от стороннего вмешательства не предусмотрена.

**Среда, передаваемая программе:**

1. Что такое среда?

Область памяти с размещенными в ней переменными средами со значениями.

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

Во время запуска операционной системы.

3. Откуда берётся информация, записываемая в среду?

Из AUTOEXEC.BAT, при запуске операционной системы.

**Выводы.**

Были проведены исследование интерфейса управляющей программы и загрузочных модулей; исследование префикса сегмента программы (PSP) и среды, передаваемой программе.

## ПРИЛОЖЕНИЕ А.

### Исходный код.

lb2.asm:

```
code SEGMENT
    ASSUME CS:code, DS:code
org 100H
begin:
    jmp opening

data:

Task1 DB 'Segment address of inaccessible memory:      ',0DH,0AH,'$'
Task2 DB 'Segment address of the environment passed to the program:
',0DH,0AH,'$'
Task3_1 DB 'Command line tail:        ',0DH,0AH,'$'
Task3_2 DB 'Line tail is absent',0DH,0AH,'$'
Task4 DB 'Environment area content:',0DH,0AH,'$'
Task5 DB 'Module path:',0DH,0AH,'$'

opening:
    call continued
    xor AL, AL
    mov AH, 4Ch
    int 21h

print_function PROC
    mov ah, 09h
    int 21h
    ret
print_function ENDP

print_function2 PROC
    push ax
    mov ah, 02h
    int 21h
    pop ax
    ret
```

```
print_function2 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
            ;байт в AL переводится в два символа
            ;шестнадцатиричного числа в AX
            push CX
            mov AH,AL
            call TETR_TO_HEX
            xchg AL,AH
            mov CL,4
            shr AL,CL
            call TETR_TO_HEX;в AL старшая цифра
            pop CX;в AH младшая
            ret
      BYTE_TO_HEX ENDP

      WRD_TO_HEX PROC
            ;перевод в 16 с/с 16-ти разрядного числа
            ;в AX - число, DI - адрес последнего символа
            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
      ;перевод в 10 с/с, SI - адрес поля младшей цифры
            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

inaccessible_memory PROC
    mov AX, DS:[2h]
    mov DI, offset Task1 + 45
    call WRD_TO_HEX
    mov DX, offset Task1
    call print_function
    ret
inaccessible_memory ENDP

address_of_environment PROC
    mov AX, DS:[2Ch]
```

```
    mov DI, offset Task2 + 60
    call WRD_TO_HEX
    mov DX, offset Task2
    call print_function
    ret
address_of_environment ENDP


command_line_tail PROC
    mov CL, DS:[80h]
    cmp CL, 0h
    je absent
    mov DX, offset Task3_1
    call print_function
    mov SI, 81h
loop1:
    mov DL, DS:[SI]
    call print_function2
    inc SI
    loop loop1

    mov DL, 0Dh
    call print_function2
    mov DL, 0Ah
    call print_function2
    ret
absent:
    mov DX, offset Task3_2
    call print_function
    ret
command_line_tail ENDP

environment_area PROC
    mov DX, offset Task4
    call print_function
    mov ES, DS:[2Ch]
    xor DI, DI
    mov AX, ES:[di]
    cmp AX, 00h
```

```
    jz ending
    add DI, 2
reading:
    mov DL, AL
    call print_function2
    mov AL, AH
    mov AH, ES:[DI]
    inc DI
    cmp AX, 00h
    jne reading
ending:
    mov DL, 0Dh
    call print_function2
    mov DL, 0Ah
    call print_function2
    mov DX, offset Task5
    call print_function
    add DI, 2
    mov DL, ES:[DI]
    inc DI
path:
    call print_function2
    mov DL, ES:[DI]
    inc DI
    cmp DL, 00h
    jne path
    ret
environment_area ENDP

continued PROC
    call inaccessible_memory
    call address_of_environment
    call command_line_tail
    call environment_area
     ret
continued ENDP

code ENDS
END begin
```