**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МОЭВМ**

# ОТЧЕТ

**по лабораторной работе №2**

**по дисциплине «Операционные системы»**

**Тема: Исследование интерфейсов программных модулей.**

Студент гр. 9383 _____ Гладких А.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

# ПОСТАНОВКА ЗАДАЧИ

**Цель работы.**

Исследование интерфейса управляющей программы и загрузочных модулей. Исследование префикса сегмента программы (PSP) и среды, передаваемой программе.

**Сведения о функциях и структурах данных управляющей программы.**

| Процедура | Описание |
|---|---|
| TETR_TO_HEX | Перевод десятичной цифры в код символа ASCII |
| BYTE_TO_HEX | Перевод байта в AL в два символа шестн. числа AX |
| WRD_TO_HEX | Перевод в 16 с/с 16-ти разрядного числа |
| BYTE_TO_DEC | Перевод в 10 с/с, SI – адрес поля младшей цифры |
| WRITEWRD | Функция печати строки на экран |
| WRITEBYTE | Функция печати символа на экран |
| ENDLINE | Функция печати символов переноса строки |
| TASK_1 | Вывод адреса недоступной памяти в шестнадцатеричном виде |
| TASK_2 | Вывод адреса среды в шестнадцатеричном виде |
| TASK_3 | Вывод хвоста командной строки в символьном виде |
| TASK_4 | Вывод содержимого области среды в символьном виде |
| TASK_5 | Вывод пути загружаемого модуля |

**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.

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

**Исходный код.**

Исходный код представлен в приложении А.

## РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ ПРОБЛЕМ

**Шаг 1.** Был написан и отлажен .COM модуль, который выводит на экран требуемую в задании информацию.


Рисунок 1. Демонстрация работы .COM модуля, хвост командной строки пуст


Рисунок 2. Демонстрация работы .COM модуля, хвост командной строки «tail test»

**Шаг 2.** По итогу написания работы были отвечены контрольные вопросы «Сегментный адрес недоступной памяти»:

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

   Ответ: на первый байт после памяти, выделенной под программу.

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

   Ответ: адрес недоступной памяти располагается в сторону увеличения адресов.

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

   Ответ: да, так как в DOS нет защиты памяти от перезаписи.

**Шаг 3.** По итогу написания работы были отвечены контрольные вопросы «Среда передаваемая программе»:

1. ***Что такое среда?***

   Ответ: область памяти, хранящая переменные среды, которые хранят информацию о состоянии системы и непосредственно влияют на поведение различных программ в компьютере.

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

   Ответ: так называемый «Master Environment» создается при загрузке системы DOS. При запуске приложения текущая среда копируется в адресное пространство запущенной программы.

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

   Ответ: в системе DOS информация берется из файла CONFIG.SYS

**Выводы.**

Были исследованы интерфейс управляющей программы и загрузочных модулей, префикс сегмента программы (PSP) и среда, передаваемая программе.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab.asm

```
TESTPC  SEGMENT
        ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
org 100h

start: jmp begin

UNAVAILABLE_MEM db 'Unavailable Memory:    ', 0DH, 0AH, '$'
ENV_SEG db 'Environment Segment:    ', 0DH, 0AH, '$'
COMMAND_LINE_ARGS db 'Command Line Args:$'
COMMAND_LINE_ARGS_EMPTY db 'Command Line Args: empty', 0DH, 0AH, '$'
ENV_SEG_CONTENT db 'Environment Segment Content:', 0DH, 0AH, '$'
BOOT_PATH db 'Boot Path:$'


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

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

WRITEWRD  PROC  NEAR
    push ax
    mov ah, 9
    int 21h
    pop ax
    ret
WRITEWRD  ENDP

WRITEBYTE  PROC  NEAR
    push ax
    mov ah, 02h
    int 21h
    pop ax
    ret
WRITEBYTE  ENDP

ENDLINE PROC NEAR
    push ax
    push dx

    mov dl, 0dh
    call WRITEBYTE

    mov dl, 0ah
    call WRITEBYTE

    pop dx
    pop ax
    ret
ENDLINE ENDP

TASK_1 PROC NEAR

    push ax
    push di
```

```
    mov ax,ds:[02h]
mov di, offset UNAVAILABLE_MEM
add di, 22
call WRD_TO_HEX

mov dx, offset UNAVAILABLE_MEM
call WRITEWRD

    pop di
    pop ax
ret


TASK_1 ENDP

TASK_2 PROC NEAR

    push ax
    push cx
    push di

    mov ax,ds:[2ch]
mov di, offset ENV_SEG
add di, 23
call WRD_TO_HEX

mov dx, offset ENV_SEG
call WRITEWRD

    pop di
    pop cx
    pop ax
ret


TASK_2 ENDP

TASK_3 PROC NEAR

    push ax
    push cx
    push dx
```

```
    push di

    xor cx, cx
    xor di, di

    mov cl, ds:[80h]
    cmp cl, 0
    je no_args

    mov dx, offset COMMAND_LINE_ARGS
    call WRITEWRD

for_loop:
    mov dl, ds:[81h + di]
    call WRITEBYTE

    inc di

    loop for_loop
    call ENDLINE
    jmp restore

no_args:
    mov dx, offset COMMAND_LINE_ARGS_EMPTY
    call WRITEWRD

restore:
    pop di
    pop dx
    pop cx
    pop ax
ret

TASK_3 ENDP

TASK_4 PROC NEAR

    push ax
    push dx
    push es
    push di
```

```
    mov dx, offset ENV_SEG_CONTENT
    call WRITEWRD

    xor di, di
    mov ax, ds:[2ch]
    mov es, ax

content_loop:
    mov dl, es:[di]
    cmp dl, 0
    je end_string2

    call WRITEBYTE

    inc di
    jmp content_loop

end_string2:
    call ENDLINE

    inc di

    mov dl, es:[di]
    cmp dl, 0
    jne content_loop

    call TASK_5

    pop di
    pop es
    pop dx
    pop ax
ret

TASK_4 ENDP

TASK_5 PROC NEAR

    push ax
    push dx
    push es
    push di
```

```
    mov dx, offset BOOT_PATH
call WRITEWRD

add di, 3

boot_loop:
mov dl, es:[di]
cmp dl,0
je restore2

call WRITEBYTE

inc di

jmp boot_loop

restore2:
    call ENDLINE

    pop di
    pop es
    pop dx
    pop ax
ret

TASK_5 ENDP

begin:

    call TASK_1
    call TASK_2
    call TASK_3
    call TASK_4

    ; Выход в DOS
    xor al, al
    mov ah, 4ch
    int 21h

TESTPC  ENDS
        END startEND Main
```