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:

Студент гр. 8383 _____ Колмыков В.Д.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Исследование интерфейса управляющей программы и загрузочных модулей а также префикса сегмента программы и среды, передаваемой программе.

Процедуры, используемые в работе.

Название процедуры	Описание
GET_LOCKED_MEMORY	Процедура вывода на экран адреса недоступной памяти
WRITE	Процедура печати строки по смещению DX
WRITE_HEX_WORD	Вывод содержимого AX в 16-иричной с.с.
WRITE_HEX_BYTE	Вывод содержимого AL в 16-ричной с.с.
GET_ENVIRONMENT	Процедура печати сегментного адреса среды
GET_TAIL	Процедура печати хвоста командной строки в символьном виде
WRITE_SYMB_BYTE	Вывод символа из AL
GET_ENVIRONMENT_CONTENT	Процедура печати содержимого среды и пути загружаемого модуля в символьном виде

Ход работы.

Был написан и отлажен программный модуль типа .COM, который распечатывает на экран следующую информацию:

1) Сегментный адрес недоступной памяти
2) Сегментный адрес среды

3) Хвост командной строки
4) Содержимое области среды
5) Путь загружаемого модуля

Код модуля приведен в приложении А. Результат работы программы представлен на рис. 1.

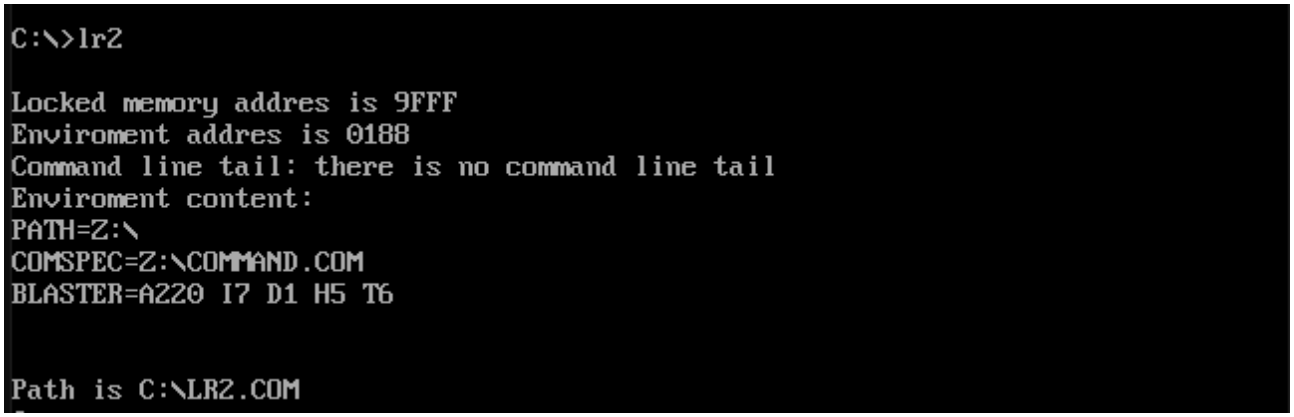

Рисунок 1 – Результат работы программы

Ответы на контрольные вопросы.

Сегментный адрес недоступной памяти

1) Адрес недоступной памяти указывает на служебную часть памяти (память, которую DOS не может выделить под программу). Сам адрес указывает на сегментный адрес последнего параграфа памяти, используемого DOS для запуска программ.
2) Недоступная память расположена сразу за областью памяти, которую DOS отводит пользовательским программам.
3) В эту область памяти можно писать, так как DOS не контролирует обращение программ к памяти.

Среда, передаваемая программе

1) Среда представляет собой последовательность символьных строк вида <имя>=<параметр>. Например, COMSPEC определяет путь к COMMAND.COM, PATH определяет пути к программным файлам, которые будут вызываться на выполнение. Каждая строка завершается байтом нулей.

2) Изначально, при запуске DOS создается корневая среда, относящаяся к COMMAND.COM. Затем, когда COMMAND.COM запускает пользовательскую программу или одна программа запускает другую создается порожденный процесс, который получает собственный экземпляр блока среды, при этом по умолчанию создается точная копия среды родителя.
3) Из родительской среды.

Выводы.

В ходе выполнения лабораторной работы был исследован интерфейс управляющей программы и загрузочных модулей, а также префикс сегмента программы (PSP) и среды, передаваемой программе.

ПРИЛОЖЕНИЕ А

```
LAB2 SEGMENT
     ASSUME  CS:LAB2, DS:LAB2, ES:NOTHING, SS:NOTHING
     ORG 100H
START: JMP BEGIN
LOCKED_MEMORY_STR db 13, 10, "Locked memory addres is $"
ENVIRONMENT db 13, 10, "Enviroment addres is $"
TAIL db 13, 10, "Command line tail: $"
NO_TAIL db "there is no command line tail$"
ENVIRONMENT_CONTENT db 13, 10, "Enviroment content:", 13, 10, '$'
ENTER_SYMB db 13, 10, '$'
PATH_STR db 13, 10, "Path is $"
;_____________________________________________________
GET_LOCKED_MEMORY PROC
     push AX
     push DX

     mov DX, offset LOCKED_MEMORY_STR
     call WRITE
     mov AX, DS:[02h]
     call WRITE_HEX_WORD

     pop DX
     pop AX
     ret
GET_LOCKED_MEMORY ENDP
;_____________________________________________________
WRITE PROC
     push AX
     mov AH, 9h
     int 21h
     pop AX
     ret
```

```
WRITE ENDP
;_____________________________________________________
WRITE_HEX_WORD PROC
      push AX

      push AX
      mov AL, AH
      call WRITE_HEX_BYTE
      pop AX
      call WRITE_HEX_BYTE

      pop AX
      ret
WRITE_HEX_WORD ENDP
;_____________________________________________________
WRITE_HEX_BYTE PROC
      push AX
      push BX
      push DX

      mov AH, 0
      mov BL, 16
      div BL
      mov DX, AX
      mov AH, 02h
      cmp DL, 0Ah
      jl PRINT
      add DL, 7
PRINT:
      add DL, '0'
      int 21h;

      mov DL, DH
      cmp DL, 0Ah
```

```
     jl PRINT2
     add DL, 7
PRINT2:
     add DL, '0'
     int 21h;

     pop DX
     pop BX
     pop AX
     ret
WRITE_HEX_BYTE ENDP
;_____________________________________________________
GET_ENVIRONMENT PROC
     push AX
     push DX

     mov DX, offset ENVIRONMENT
     call WRITE
     mov AX, DS:[2Ch]
     call WRITE_HEX_WORD

     pop DX
     pop AX
     ret
GET_ENVIRONMENT ENDP
;_____________________________________________________
GET_TAIL PROC
     push AX
     push CX
     push DX
     push SI

     mov DX, offset TAIL
     call WRITE
```

```
	xor CX, CX
	mov CL, DS:[80h]
	cmp CL, 0
	jne REWRITING_TAIL
	mov DX, offset NO_TAIL
	call WRITE
	jmp END_OF_PROC_TAIL
REWRITING_TAIL:
	xor SI, SI
	xor AX, AX
CYCLE:
	mov AL, DS:[81h + SI]
	call WRITE_SYMB_BYTE
	inc SI
	loop CYCLE

END_OF_PROC_TAIL:
	pop SI
	pop DX
	pop CX
	pop AX
	ret
GET_TAIL ENDP
;_____________________________________________________
WRITE_SYMB_BYTE PROC
	push AX
	push DX

	xor DX, DX
	mov DL, AL
	mov AH, 02h
	int 21h

	pop DX
```

```
      pop AX
      ret
WRITE_SYMB_BYTE ENDP
;_____________________________________________________
GET_ENVIRONMENT_CONTENT PROC
      push AX
      push BX
      push DX
      push ES
      push SI

      mov DX, offset ENVIRONMENT_CONTENT
      call WRITE
      xor SI, SI
      mov BX, 2Ch
      mov ES, [BX]
READING_STR:
      cmp BYTE PTR ES:[SI], 0h
      je NEW_LINE
      mov AL, ES:[SI]
      call WRITE_SYMB_BYTE
      jmp CHECK_END
NEW_LINE:
      mov DX, offset ENTER_SYMB
      call WRITE
CHECK_END:
      inc SI
      cmp WORD PTR ES:[SI], 0001h
      je PATH
      jmp READING_STR
PATH:
      mov DX, offset PATH_STR
      call WRITE
      add SI, 2
```

```
CYCLE_PATH:
     cmp BYTE PTR ES:[SI], 00h
     je END_OF_PROC_CONTENT
     mov AL, ES:[SI]
     call WRITE_SYMB_BYTE
     inc SI
     jmp CYCLE_PATH

END_OF_PROC_CONTENT:
     pop SI
     pop ES
     pop DX
     pop BX
     pop AX
     ret
GET_ENVIRONMENT_CONTENT ENDP
;_____________________________________________________
BEGIN:
     call GET_LOCKED_MEMORY
     call GET_ENVIRONMENT
     call GET_TAIL
     call GET_ENVIRONMENT_CONTENT

     ;TO DOS
     xor AL, AL
     mov AH, 4Ch
     int 21h
LAB2 ENDS
END START
```