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 _____ Санников 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 стоит префикс сегмента программы (PSP) и помещает его адрес в сегментный регистр. Исследование префикса среды программы (PSP) и среды, передаваемой программе.

Задание.

1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .COM, который выбирает и распечатывает следующую информацию:

- Сегментный адрес недоступной памяти, взятый из PSP, в шестнадцатеричном виде;

- Сегментный адрес среды, передаваемой программе, в шестнадцатеричном виде;

- Хвост командной строки в символьном виде;

- Содержимое области среды в символьном виде;

- Путь загружаемого модуля.

Сохраните результаты, полученные программой, и включите их в отчет.

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

Ход работы.

Для выполнение лабораторной работы был написан .COM модуль, в котором реализованы следующие процедуры:

1) UM_FUNC — процедура для вывода сегментного адреса недоступной памяти, взятого из PSP, в шестнадцатеричном виде. В AX кладем то, что располагается по адресу DS:[2h] и переводим в шестнадцатеричный вид.

2) EA_FUNC — процедура для вывода сегментного адреса среды, передаваемой программе, в 16-ричном виде. В AX кладем то, что располагается по адресу DS:[2Ch] и переводим в шестнадцатеричный вид.

3) CLT_FUNC — процедура нужна для вывода хвоста командной строки посимвольно. Если строка пустая, то будет выведено соответствующее сообщение (проверка идет по адресу DS:[80h])

4) CEA_FUNC — процедура, которая отвечает за вывод информации о среде в символьном виде и пути загружаемого модуля. В ES кладем адрес DS:[2Ch], далее через увеличение DI выводим посимвольно информацию, пока не дойдем до байта нуля (в DL будет значение 0h). Далее увеличиваем DI на 3, чтобы получить начало маршрута загруженной программы. Выводим его тоже посимвольно до 0h в DL.

5) PRINT_SYMB — вспомогательная процедура вывода символа, используется в процедуре CEA_FUNC для вывода данных о среде и пути загружаемого модуля.

Результаты работы программы см на рисунках 1 и 2.

Рисунок 1 — Результат работы программы без «хвоста» командной строки.

Рисунок 2 — Результат работы программы с «хвостом» командной строки.

Исходный код программы см в приложении А.

Ответы на контрольные вопросы.

1. Сегментный адрес недоступной памяти:

- На какую область памяти указывает адрес недоступной памяти?

На сегмент, расположенный сразу после выделенной программе памяти.

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

В первом байте памяти, расположенном после выделенной программе памяти. Найти этот адрес можно в PSP по смещению 2h.

- Можно ли в эту область памяти писать?

В DOS нет защиты памяти от перезаписи => можно

2. Среда, передаваемая программе:

- Что такое среда?

Среда – это область памяти, содержащая последовательность символьных строк вида “имя = параметр”, в которых записана информация переменных окружений. Есть несколько переменных окружений, например, PATH, в которое записан путь к каталогу, где система ищет исполняемый файл.

- Когда создается среда? Перед запуском приложения или в другое время?

Изначально среда создается при запуске ОС. Когда же запускается приложение, создается копия этой среды, в которую добавляются дополнительные параметры для данного приложения, если это требуется.

- Откуда берется информация, записываемая в среду?

В большинстве случаев переменные окружения определяются с помощью команды SET в файле Autoexec.bat. За исключением CONFIG — он в файле Config.sys, PROMPT — определяется отдельной командой оболочки DOS и PATH — задается отдельно в файле Autoexec.bat.

Вывод.

В ходе данной лабораторной работы был написан текст .COM модуля. Был исследован интерфейс управляющей программы и загрузочных модулей, а также исследован префикс среды программы (PSP) и среды, передаваемой программе. Была написана программа, выводящая в консоль информацию согласно заданию.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл lab2.asm

```
TESTPC SEGMENT
     ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
     ORG  100H
START:    jmp BEGIN
;------------------------------
UM db 'Unavailable memory:     h', 0DH, 0AH, '$'
EA db 'Address of the environment:     h', 0DH, 0AH, '$'
CLT db 'Command line tail: ', '$'
EMP db 'Command line tail is empty', 0DH, 0AH, '$'
CEA db 'Contents of the environment area:  ', 0DH, 0AH,'$'
PTH db 'The path of the load module:  ', 0DH, 0AH, '$'
;------------------------------
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
; байт в AL переводится в два символа 16-го 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 near
; перевод в 16 с/с 16-ти разрядного числа
; в AX - число, в 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, в 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_1
      or AL, 30h
      mov [SI], AL
end_1:  pop DX
      pop CX
      ret
BYTE_TO_DEC ENDP
;----------------------
PRINT PROC near
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
PRINT ENDP
;----------------------
PRINT_SYMB PROC near
      push AX
      mov AH, 02h
      int 21h
      pop AX
      ret
PRINT_SYMB ENDP
;----------------------
UM_FUNC PROC near
      mov AX, DS:[2h]
      mov DI, offset UM
      add DI, 23
      call WRD_TO_HEX
      mov DX, offset UM
```

```
      call PRINT
      ret
UM_FUNC ENDP
;---------------------------------
EA_FUNC PROC near
      mov AX, DS:[2Ch]
      mov DI, offset EA
      add DI, 31
      call WRD_TO_HEX
      mov DX, offset EA
      call PRINT
      ret
EA_FUNC ENDP
;---------------------------------
CLT_FUNC PROC near
      xor CX, CX
      mov CL, DS:[80h]
      cmp CL, 0h
      je if_empty
      mov DX, offset CLT
      call PRINT
      mov SI, 81h
loop_clt:
      mov DL, DS:[SI]
      call PRINT_SYMB
      inc SI
      loop loop_clt

      mov DL, 0Dh
      call PRINT_SYMB
      mov DL, 0Ah
      call PRINT_SYMB
      ret
if_empty:
      mov DX, offset EMP
      call PRINT
      ret
CLT_FUNC ENDP
;---------------------------------
CEA_FUNC PROC near
      mov DX, offset CEA
      call PRINT
      mov ES, DS:[2Ch]
      xor DI, DI
print1:
      mov DL, ES:[DI]
      cmp DL, 0h
      je print2
      call PRINT_SYMB
      inc DI
      jmp print1
print2:
```

```
      mov DL, 0Dh
      call PRINT_SYMB
      mov DL, 0Ah
      call PRINT_SYMB
      inc DI
      mov DL, ES:[DI]
      cmp DL, 0h
      jne print1

      mov DX, offset PTH
      call PRINT
      add DI, 3
print3:
      mov DL, ES:[DI]
      cmp DL, 0h
      je end_print
      call PRINT_SYMB
      inc DI
      jmp print3
end_print:
      ret
CEA_FUNC ENDP
;-----------------------------------
BEGIN:
      call UM_FUNC
      call EA_FUNC
      call CLT_FUNC
      call CEA_FUNC
      xor AL, AL
      mov AH, 4Ch
      int 21h
TESTPC ENDS
      END START
```