**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра математического обеспечения и применения ЭВМ**

# ОТЧЕТ

**по практической работе №6**
**по дисциплине «Операционные системы»**
**Тема: Построение модуля динамической структуры**

Студент гр. 8381 _____ Киреев К.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

**Цель работы.**

Исследование возможности построения загрузочного модуля динамической структуры. В отличии от предыдущих лабораторных работ в этой работе рассматривается приложение, состоящее из нескольких модулей, а не из одного модуля простой структуры. В этом случае разумно предположить, что все модули приложения находятся в одном каталоге и полный путь в этот каталог можно взять из среды, как это делалось в работе 2. Понятно, что такое приложение должно запускаться в соответствии со стандартами ОС.

В работе исследуется интерфейс между вызывающим и вызываемым модулями по управлению и по данным. Для запуска вызываемого модуля используется функция 4B00h прерывания int 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

**Выполнение работы.**

Написан текст исходного EXE модуля, который выполняет следующие функции. Подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится он сам. Вызываемому модулю передается новая среда, созданная вызывающим модулем и новая командная строка. Вызываемый модуль запускается с использованием загрузчика. После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы. Необходимо проверять причину завершения и, в зависимости от значения, выводить соответствующее сообщение. Если причина завершения 0, то выводится код завершения. В качестве вызываемой программы была взята программа ЛР 2, которая распечатывает среду и командную строку.

Полученный модуль был отлажен. Далее отлаженная программа была запущена, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается,

ожидая символ с клавиатуры. Введен символ ‘t’. Результаты выполнения программы представлены на рис. 1.

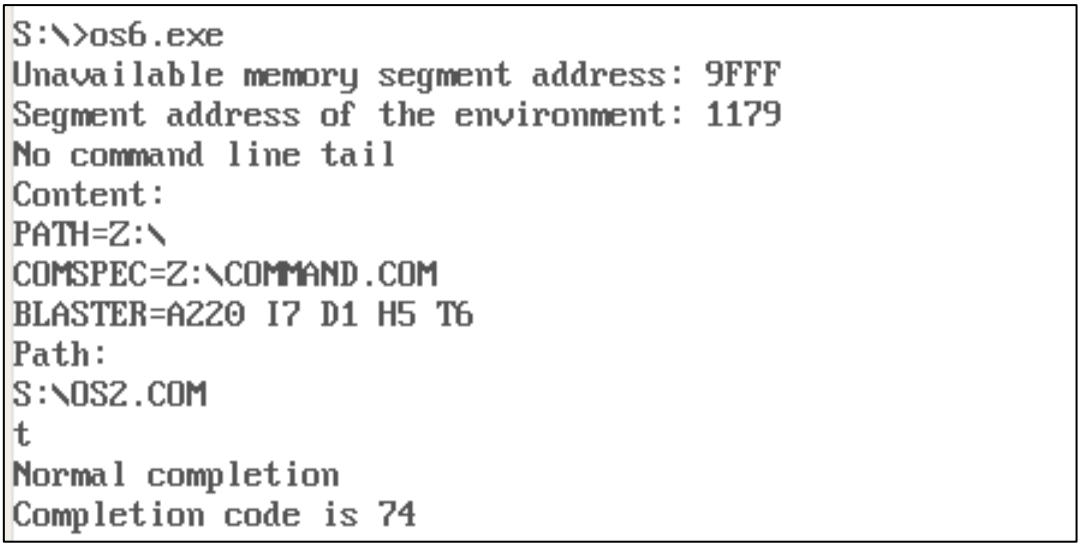


Рисунок 1 – Результат выполнения OS6.EXE, когда текущим каталогом является каталог с разработанными модулями.

Далее программа была запущена, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Введена комбинация Ctrl-C. Результаты выполнения представлены на рис. 2.

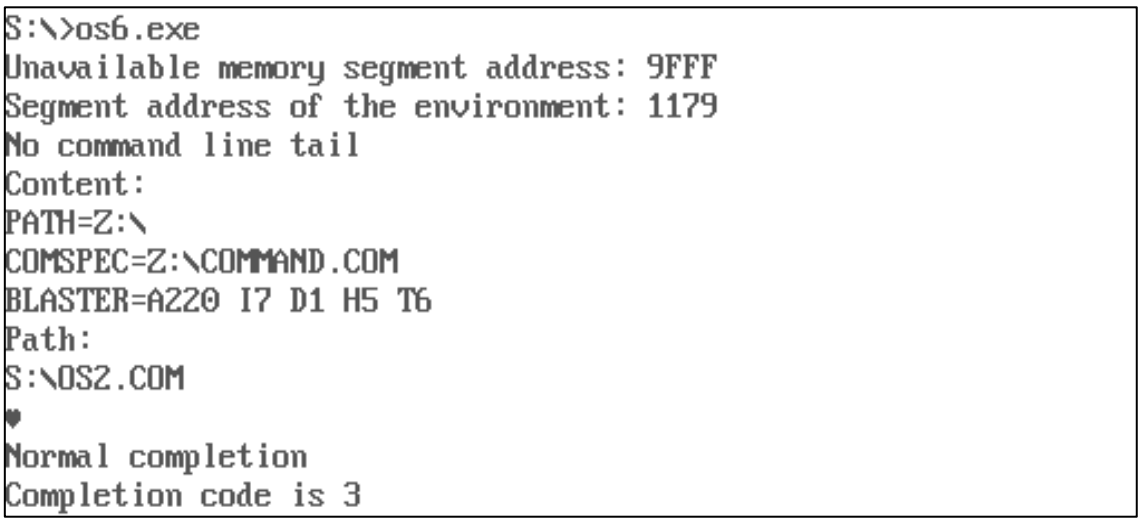


Рисунок 2 – Результат выполнения OS6.EXE, когда текущим каталогом является каталог с разработанными модулями и введена комбинация Ctrl-C.

Далее отлаженная программа была запущена, когда оба модуля находятся не в текущем каталоге. Введен символ ‘t’. Результаты выполнения программы представлены на рис. 3.

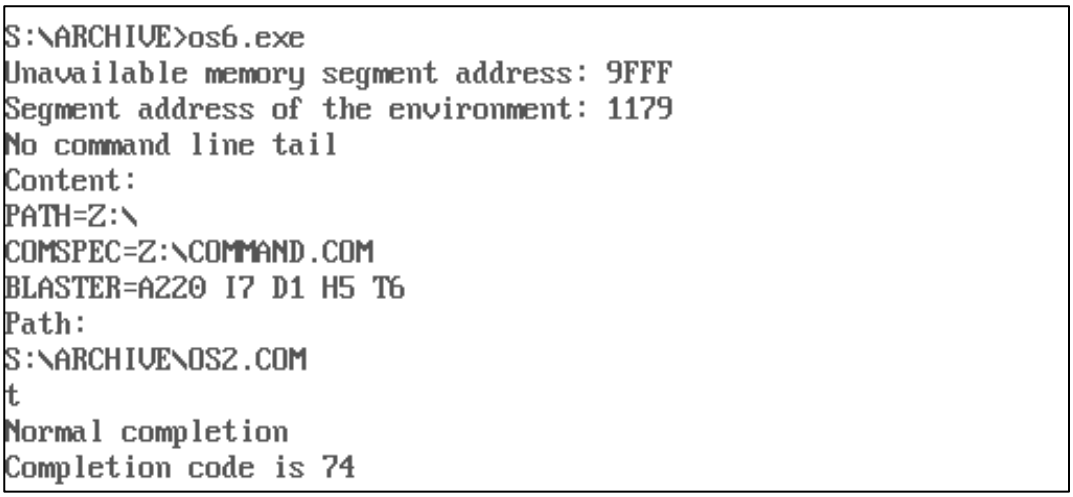


Рисунок 3 – Повторный запуск программы, когда оба модуля не в текущем каталоге и введен символ ‘t’.

Далее отлаженная программа была запущена, когда оба модуля находятся не в текущем каталоге. Введена комбинация Ctrl-C. Результаты выполнения представлены на рис. 4.

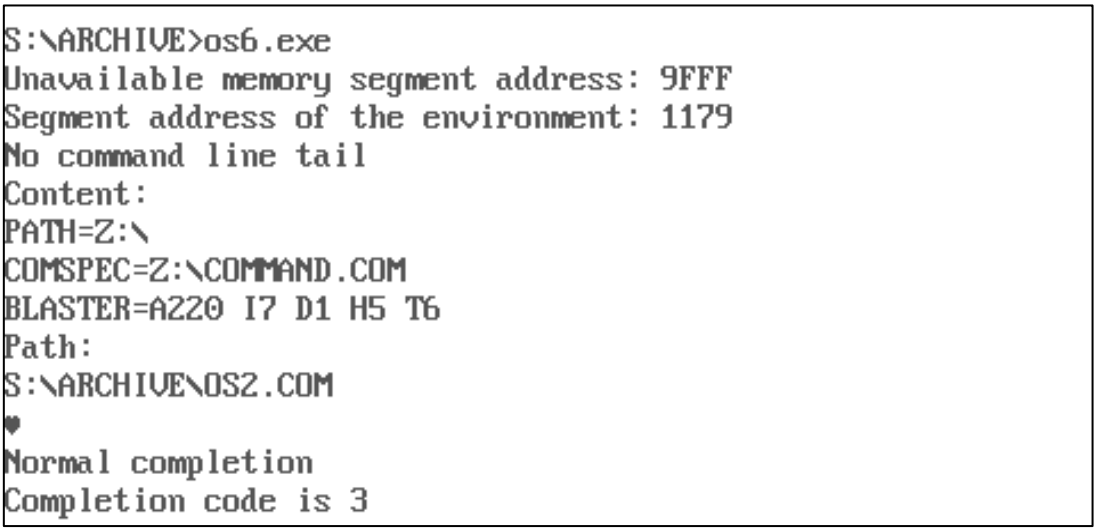


Рисунок 4 – Повторный запуск программы, когда оба модуля не в текущем каталоге и введена комбинация Ctrl-C.

Далее отлаженная программа была запущена, когда модули находятся в разных каталогах. Результаты выполнения представлены на рис. 5.

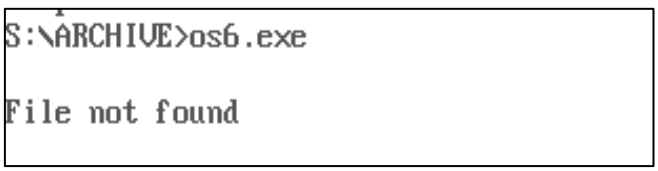


Рисунок 5 – Запуск программы, когда модули находятся в разных каталогах.

**Контрольные вопросы**

- **Как реализовано прерывание Ctrl-C?**

Большая часть функций ввода-вывода из диапазона 01H..0CH проверяют перед своим выполнением наличие в кольцевом буфере клавиатуры кода 03 (Ctrl+C) и при обнаружении этого кода выполняют команду INT 23H. В векторе 23H обычно находится адрес программы DOS, завершающей текущий процесс. Исключение составляют функции 06H и 07H, нечувствительные к Ctrl+C, а также функции 02H и 09H, которые анализируют кольцевой буфер на предмет наличия там Ctrl+C один раз на каждые 64 вызова. Далее адрес по вектору INT 23H копируется в поле PSP Ctrl-Break Address функциями 26H и 4CH. Проверка на Ctrl+C осуществляется независимо от перенаправления ввода-вывода, а также независимо от состояния системного флага BREAK.

- **В какой точке заканчивается вызываемая программа, если код причины завершения 0?**

Вызываемая программа заканчивается в месте вызова функции 4CH прерывания INT 21H.

- **В какой точке заканчивается вызываемая программа по прерыванию Ctrl-C?**

Вызываемая программа заканчивается в месте, где она ожидала ввода символа - в точке вызова функции 01H прерывания INT 21H.

**Вывод.**

В процессе выполнения данной лабораторной работы была исследована возможность построения загрузочного модуля динамической структуры.

# ПРИЛОЖЕНИЕ А

# ИСХОДНЫЙ КОД ПРОГРАММЫ. OS6.ASM

```
Astack segment stack
    dw 256 dup(?)
Astack ends

data segment
     filename db 'OS2.COM', 0
     parameters dw 7 dup(0)
     keep_ss dw 0
     keep_sp dw 0
     dos_4ah_err7   db   13,10,'Memory   control   block   is
destroyed',13, 10,'$'
     dos_4ah_err8  db  13,10,'Not  enough  memory  to  execute
function',13, 10,'$'
     dos_4ah_err9 db 13,10,'Invalid memory block address',13,
10,'$'

     dos_4bh_err1  db  13,10,'Function  number  is  invalid',13,
10,'$'
     dos_4bh_err2 db 13,10,'File not found',13, 10,'$'
     dos_4bh_err5 db 13,10,'Disk error',13, 10,'$'
     dos_4bh_err8 db 13,10,'Insufficient memory',13, 10,'$'
     dos_4bh_err10  db  13,10,'Wrong  environment  string',13,
10,'$'
     dos_4bh_err11 db 13,10,'Wrong format',13, 10,'$'

     dos_4dh_err0 db 13,10,'Normal completion$'
     dos_4dh_err1 db 13,10,'Completion by Ctrl-Break$'
     dos_4dh_err2 db 13,10,'Device error termination$'
     dos_4dh_err3 db 13,10,'Completion by function 31h$'

     mesto db 16 dup (0)
     child_path db 64 dup (0),'$'
     completion_code db 13,10,'Completion code is       $'
data ends
```

```
code segment
    assume cs:code, ds:data, ss:Astack

pushall macro
     irp case, <ax,bx,cx,dx,si,di,es,ds,sp,bp>
          push &case&
     endm
endm
popall macro
     irp case, <bp,sp,ds,es,di,si,dx,cx,bx,ax>
          pop &case&
     endm
endm

print proc near
     push ax
     mov ah, 09h
     int 21h
     pop ax
     ret
print endp

word_to_str proc near
     ;на входе ax число 16 бит
     ;si указатель на строку
     ;bx разрядность результата
     pushall
    cmp bx, 16
    ja end_wts
    cmp ax, 7FFFh
    jna plus
    mov byte ptr [si], '-'
    inc si
    not ax
    inc ax
    plus:
     xor cx, cx
     jmp manipulation
```

```
    manipulation:
     xor dx, dx
          div bx
          mov di, ax
          mov al, dl
          cmp al, 10
          sbb al, 69h
          das
          push di
          lea di, mesto
          add di, cx
          mov byte ptr [di], al
          pop di
          mov ax, di
    inc cx
    test ax, ax
    jz endrep
    jmp manipulation
    endrep:
     lea di, mesto
          add di, cx
    copyrep:
          dec di
          mov dl, byte ptr [di]
          mov byte ptr [si], dl
          inc si
          loop copyrep
    end_wts:
    popall
    ret
word_to_str endp

prep proc near
     pushall
     mov bx, offset zseg
     mov ax, es
     sub bx, ax
     mov cl, 4
```

```
     shr bx, cl
     mov ah, 4Ah
     int 21h
     jnc exec_dos_4ah
     irpc case, 789
          cmp ax, &case&
          je type_dos_4ah_err&case&
     endm
     irpc case, 789
     type_dos_4ah_err&case&:
          lea dx, dos_4ah_err&case&
          call print
          mov ax, 4C00h
          int 21h
     endm
     exec_dos_4ah:
     mov ax, es
     xor si, si
     irp case, <0, ax, 81h, ax, 5Ch, ax, 6Ch>
          mov [parameters+si], &case&
          add si, 2
     endm
     popall
     ret
prep endp

load proc near
     pushall
     mov es, es:[2Ch]
     xor si, si
     lea di, child_path
     skip_env:
          mov dl, es:[si]
          cmp dl, 00
          je end_env
          inc si
          jmp skip_env
     end_env:
```

```
     inc si
     mov dl, es:[si]
     cmp dl, 00
     jne skip_env
     add si, 3
path_change:
     mov dl, es:[si]
     cmp dl, 00
     je final_name
     mov [di], dl
     inc si
     inc di
     jmp path_change
final_name:
xor si,si
child_name:
     mov dl, byte ptr [filename+si]
     mov byte ptr [di-7], dl
     inc di
     inc si
     test dl, dl
     jne child_name
mov keep_ss, ss
mov keep_sp, sp
push ds
pop es
lea bx, parameters
lea dx, child_path
mov ah, 4bh
mov al, 0
int 21h
jnc exec_dos_4bh
mov bx, DATA
mov ds, bx
mov ss, keep_ss
mov sp, keep_sp

irp case, <1,2,5,8,10,11>
```

```
          cmp ax, &case&
          je type_dos_4bh_err&case&
     endm
     irp case, <1,2,5,8,10,11>
     type_dos_4bh_err&case&:
          lea dx, dos_4bh_err&case&
          call print
          mov ax, 4C00h
          int 21h
     endm

     exec_dos_4bh:
     mov ax, 4D00h ;ah - причина, al- код завершения
     int 21h
     irpc case, 0123
          cmp ah, &case&
          je type_dos_4dh_err&case&
     endm
     irpc case, 0123
     type_dos_4dh_err&case&:
          lea dx, dos_4dh_err&case&
          call print
          jmp exec_dos_4dh
     endm

     exec_dos_4dh:
          xor ah, ah
          lea si, completion_code+21
          mov bx, 16
          call word_to_str
          lea dx, completion_code
          call print
     popall
     ret
load endp

main proc far
    push ds
```

```
    xor ax, ax
    push ax
    mov ax, DATA
    mov ds, ax
     call prep
     call load
     mov ax, 4C00h
     int 21h
     ret
main endp
code ends
zseg segment
zseg ends
end main
```