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

ОТЧЕТ

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

Студент гр. 9383 _____ Моисейченко К. А.

Преподаватель _____ Ефремов М. А.

Санкт-Петербург
2021

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.

Задание.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции:

1) 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.

2) Вызываемый модуль запускается с использованием загрузчика.

3) 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 необходимо взять программу ЛР 2, которая распечатывает среду и командную строку. Эту программу следует немного модифицировать, вставив перед выходом из нее обращение к функции ввода символа с клавиатуры. Введенное значение записывается в регистр AL и затем происходит обращение к функции выхода 4Ch прерывания int 21h.

Шаг 2. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите произвольный символ из числа A-Z. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

Шаг 3. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите комбинацию символов Ctrl-C. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

Шаг 4. Запустите отлаженную программу, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

Повторите ввод комбинаций клавиш. Занесите полученные данные в отчет.

Шаг 5. Запустите отлаженную программу, когда модули находятся в разных каталогах. Занесите полученные данные в отчет.

Основные теоретические положения.

Для загрузки и выполнения одной программы из другой используется функция 4B00h прерывания int 21h (загрузчик ОС). Перед обращением к этой функции необходимо выполнить следующие действия:

1) Подготовить место в памяти. При начальном запуске программы ей отводится вся доступная в данный момент память OS, поэтому необходимо освободить место в памяти. Для этого можно использовать функцию 4Ah прерывания int 21h. Эта функция позволяет уменьшить отведенный программе блок памяти. Перед вызовом функции надо определить объем памяти, необходимый программе ЛР6 и задать в регистре BX число параграфов, которые будут выделяться программе. Если функция 4Ah не может быть выполнена, то устанавливается флаг переноса CF=1 и в AX заносится код ошибки:

7 - разрушен управляющий блок памяти;

8 - недостаточно памяти для выполнения функции;

9 - неверный адрес блока памяти.

Поэтому после выполнения каждого прерывания int 21h следует проверять флаг переноса CF=1.

2) Создать блок параметров. Блок параметров - это 14-байтовый блок памяти, в который помещается следующая информация:

```
dw сегментный адрес среды
dd сегмент и смещение командной строки
dd сегмент и смещение первого FCB
dd сегмент и смещение второго FCB
```

Если сегментный адрес среды 0, то вызываемая программа наследует среду вызывающей программы. В противном случае вызывающая программа должна сформировать область памяти в качестве среды, начинающуюся с адреса кратного 16 и поместить этот адрес в блок параметров.

Командная строка записывается в следующем формате:

первый байт - счетчик, содержащий число символов в командной строке, затем сама командная строка, содержащая не более 128 символов.

На блок параметров перед загрузкой вызываемой программы должны указывать ES:BX.

3) Подготовить строку, содержащую путь и имя вызываемой программы. В конце строки должен стоять код ASCII 0. На подготовленную строку должны указывать DS:DX.

4) Сохранить содержимое регистров SS и SP в переменных. При восстановлении SS и SP нужно учитывать, что DS необходимо также восстановить.

Когда вся подготовка выполнена, вызывается загрузчик OS следующей последовательностью команд:

```
mov AX,4B00h
int 21h
```

Если вызываемая программа не была загружена, то устанавливается флаг переноса CF=1 и в AX заносится код ошибки:

- 1 - если номер функции неверен;
- 2 - если файл не найден;
- 5 - при ошибке диска;
- 8 - при недостаточном объеме памяти;
- 10 - при неправильной строке среды;
- 11 - если не верен формат.

Если CF=0, то вызываемая программа выполнена и следует обработать ее завершение. Для этого необходимо воспользоваться функцией 4Dh прерывания int 21h. В качестве результата функция возвращает в регистре AH причину, а в регистре AL код завершения.

Причина завершения в регистре AH представляется следующими кодами:

- 0 - нормальное завершение;
- 1 - завершение по Ctrl-Break;

- 2 - завершение по ошибке устройства;
- 3 - завершение по функции 31h, оставляющей программу резидентной.

Код завершения формируется вызываемой программой в регистре AL перед выходом в OS с помощью функции 4Ch прерывания int 21h.

В качестве вызываемой программы целесообразно использовать программу, разработанную в Лабораторной работе №2, модифицировав ее следующим образом. Перед выходом из программы перед выполнением функции 4Ch прерывания int 21h следует запросить с клавиатуры символ и поместить введенный символ в регистр AL, в качестве кода завершения. Это можно сделать с помощью функции 01h прерывания int 21h.

```
mov AH,01h
int 21h
```

Введенный символ остается в регистре AL и служит аргументом для функции 4Ch прерывания int 21h.

Контрольные вопросы по лабораторной работе.

1) Как реализовано прерывание Ctrl-C?

2) В какой точке заканчивается вызываемая программа, если код причины завершения 0?

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

Выполнение работы.

Шаг 1.

Был написан и отлажен программный модуль типа .EXE, который выполняет поставленные в задании функции.

Шаг 2.

Отлаженная программа была запущена из каталога с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Причина завершения .EXE-модуля и код завершения отражены на рисунке 1.

Рисунок 1 - Работа программы при вводе символа k.

Шаг 3.

Отлаженная программа была запущена из каталога с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Была введена комбинация символов Ctrl-C. Причина завершения .EXE-модуля и код завершения отражены на рисунке 2.

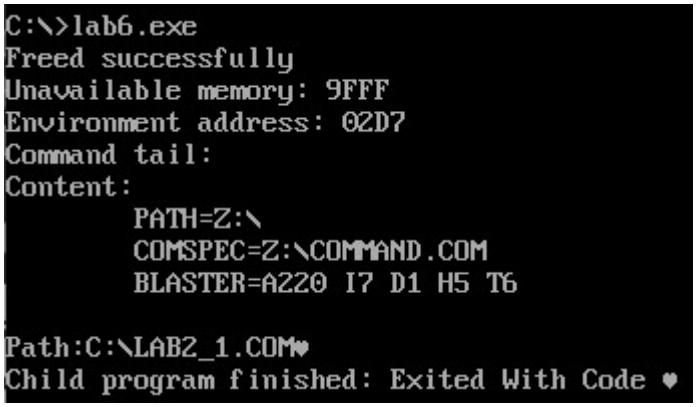

Рисунок 2 - Работа программы при вводе комбинации Ctrl+C.

Шаг 4.

Запустите отлаженную программу, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

Ввод комбинаций клавиш был повторён. Причины завершения .EXE-модуля отражены на рисунках 3-4.

Рисунок 3 - Работа программы при вводе символа k.

Рисунок 4 - Работа программы при вводе комбинации Ctrl+C.

Шаг 5.

Отлаженная программа была запущена, когда модули находятся в разных каталогах. Вывод программы представлен на рисунке 5.

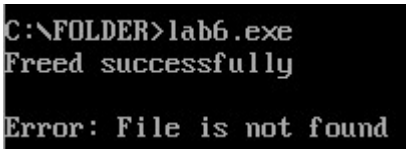

Рисунок 5 - Работа программы, запущенной из каталога, в котором лежит лишь .EXE-модуль.

Ответы на контрольные вопросы.

1. Как реализовано прерывание Ctrl-C?

При нажатии комбинации клавиш Ctrl+C срабатывает прерывание int 23h, управление передается по адресу 0000:008C. Этот адрес копируется в поле PSP

функциями DOS: 26h, которая создает PSP, и 4Ch. Исходное значение адреса обработчика Ctrl+C восстанавливается из PSP при завершении программы.

2. В какой точке заканчивается вызываемая программа, если код причины завершения 0?

   Программа завершится при обработке прерывания 4ch int 21h

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

Программа завершится в той точке, где была считана комбинация клавиш Ctrl+C/ В разработанной программе это место выхова фунции 01h прерывания int 21h.

Выводы.

Были исследованы возможности построения загрузочного модуля динамической структуры и интерфейс взаимодействия между вызывающим и вызываемым модулями по управлению и по данным. Было написано и отлажено приложение, состоящее из нескольких модулей. Программа была запущена в различных ситуациях.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab6.asm

```
ASTACK SEGMENT STACK
   DW 256 DUP(?)
ASTACK ENDS

DATA SEGMENT
    exec_param_block dw 0
    cmd_off dw 0 ; сегмент командной строки
    cmd_seg dw 0 ; смещение командной строки
    fcb1 dd 0 ; сегмент и смещение первого FCB
    fcb2 dd 0 ; сегмент и смещение второго FCB

    child_cmd_line db 1h, 0dh
    second_module_name db 'lab2_1.com', 0h
    second_module_path db 128 DUP(0)

    keep_ss dw 0
    keep_sp dw 0

    error_mem_free db 0
    mcb_crash_string db 'Error: Memory Control Block has crashed',
0DH, 0AH, '$'
    not_enough_memory_string db 'Error: Not Enough Memory', 0DH, 0AH,
'$'
    wrong_address_string db 'Error: Wrong Address', 0DH, 0AH, '$'
    free_without_error_string db 'Freed successfully', 0DH, 0AH, '$'
    child_error_function_number db 'Error: Function number is
incorrect', 0DH, 0AH, '$'
    child_error_file_not_found db 'Error: File is not found', 0DH,
0AH, '$'
    child_error_disk_error db 'Error: Disk error', 0DH, 0AH, '$'
    child_error_not_enough_mem db 'Error: Not enough memory', 0DH,
0AH, '$'
    child_error_path_string db 'Error: Path param error', 0DH, 0AH,
'$'
    child_error_wrong_format db 'Error: Wrong Format', 0DH, 0AH, '$'
    child_std_exit db 'Child program finished: Exited With Code   ',
0DH, 0AH, '$'
    child_ctrl_exit db 'Child program finished: Ctrl+Break Exit',
0DH, 0AH, '$'
    child_device_error_exit db 'Child program finished: Device Error
Exit', 0DH, 0AH, '$'
    child_int31h_exit db 'Child program finished: became resident,
int 31h Exit', 0DH, 0AH, '$'


    data_end db 0
DATA ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:ASTACK
```

```
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

FREE_UNUSED_MEMORY PROC FAR
    push ax
    push bx
    push cx
    push dx
    push es
    xor dx, dx
    mov error_mem_free, 0h
    mov ax, offset data_end
    mov bx, offset lafin
    add ax, bx
    mov bx, 10h
    div bx
    add ax, 100h
    mov bx, ax
    xor ax, ax
    mov ah, 4ah
    int 21h
    jnc free_without_error
mov error_mem_free, 1h
;mcb crash
    cmp ax, 7
    jne not_enough_memory
    mov dx, offset mcb_crash_string
    call WRITEWRD
    jmp free_unused_memory_end
```

```
not_enough_memory:
    cmp ax, 8
    jne wrong_address
    mov dx, offset not_enough_memory_string
    call WRITEWRD
    jmp free_unused_memory_end

wrong_address:
    cmp ax, 9
    jne free_unused_memory_end
    mov dx, offset wrong_address_string
    call WRITEWRD
    jmp free_unused_memory_end

free_without_error:
    mov dx, offset free_without_error_string
    call WRITEWRD

free_unused_memory_end:
    pop es
    pop dx
    pop cx
    pop bx
    pop ax
    ret

FREE_UNUSED_MEMORY ENDP


LOAD_MODULE PROC FAR
    push ax
    push bx
    push cx
    push dx
    push ds
    push es
    mov keep_sp, sp
    mov keep_ss, ss
    call GET_PATH
    mov ax, data
    mov es, ax
    mov bx, offset exec_param_block
    mov dx, offset child_cmd_line
    mov cmd_off, dx
    mov cmd_seg, ds
    mov dx, offset second_module_path
    mov ax, 4b00h
    int 21h
    mov ss, keep_ss
    mov sp, keep_sp
    pop es
    pop ds

    call ENDLINE
```

```
  jnc loaded_successfully

;function number error
    cmp ax, 1
jne load_file_not_found
mov dx, offset child_error_function_number
call WRITEWRD
jmp load_module_end

load_file_not_found:
    cmp ax, 2
jne load_disk_error
mov dx, offset child_error_file_not_found
call WRITEWRD
jmp load_module_end

load_disk_error:
    cmp ax, 5
jne load_not_enough_memory
mov dx, offset child_error_disk_error
call WRITEWRD
jmp load_module_end

load_not_enough_memory:
    cmp ax, 8
jne load_path_error
mov dx, offset child_error_disk_error
call WRITEWRD
jmp load_module_end

load_path_error:
    cmp ax, 10
jne load_wrong_format
mov dx, offset child_error_path_string
call WRITEWRD
jmp load_module_end
load_wrong_format:
    cmp ax, 11
jne load_module_end
mov dx, offset child_error_wrong_format
call WRITEWRD
jmp load_module_end

loaded_successfully:
    mov ax, 4d00h
int 21h
;std_exit
    cmp ah, 0
jne ctrl_exit
mov di, offset child_std_exit
    add di, 41
    mov [di], al
    mov dx, offset child_std_exit
call WRITEWRD
```

```
jmp load_module_end

ctrl_exit:
    cmp ah, 1
jne device_error_exit
mov dx, offset child_ctrl_exit
call WRITEWRD
jmp load_module_end

device_error_exit:
    cmp ah, 2
jne int31h_exit
mov dx, offset child_device_error_exit
call WRITEWRD
jmp load_module_end

int31h_exit:
    cmp ah, 3
jne load_module_end
mov dx, offset child_int31h_exit
call WRITEWRD
jmp load_module_end

load_module_end:
    pop dx
    pop cx
    pop bx
    pop ax
    ret

LOAD_MODULE ENDP

GET_PATH PROC NEAR

    push ax
    push dx
    push es
    push di
    xor di, di
    mov ax, es:[2ch]
    mov es, ax

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

end_string2:

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

```
    call PARSE_PATH
    pop di
    pop es
    pop dx
    pop ax
    ret

GET_PATH ENDP

PARSE_PATH PROC NEAR

    push ax
    push bx
    push bp
    push dx
    push es
    push di
    mov bx, offset second_module_path
    add di, 3

boot_loop:
    mov dl, es:[di]
    mov [bx], dl
    cmp dl, '.'
    je parse_to_slash
    inc di
    inc bx
    jmp boot_loop

parse_to_slash:
    mov dl, [bx]
    cmp dl, '\'
    je get_second_module_name
    mov dl, 0h
    mov [bx], dl
    dec bx
    jmp parse_to_slash
get_second_module_name:
    mov di, offset second_module_name
    inc bx

add_second_module_name:
    mov dl, [di]
    cmp dl, 0h
    je parse_path_end
    mov [bx], dl
    inc bx
    inc di
    jmp add_second_module_name

parse_path_end:
    mov [bx], dl

    pop di
    pop es
```

```
        pop dx
        pop bp
        pop bx
        pop ax
        ret

    PARSE_PATH ENDP

    MAIN PROC FAR
        mov ax, data
        mov ds, ax
        call FREE_UNUSED_MEMORY
        cmp error_mem_free, 0h
        jne main_end
        call GET_PATH
        call LOAD_MODULE

    main_end:
        xor al, al
        mov ah, 4ch
        int 21h

    MAIN ENDP

lafin:
CODE ENDS

END MAIN
```