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. 9382 _____ Павлов Р.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

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 4B00hпрерывания int 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Функции и структуры данных:

Название процедуры	**Описание процедуры**
main	Вызов процедур нахождения пути, выделения памяти и вызова подпрограммы
free	Процедура освобождения памяти, не занятой родительской программой
find_path	Процедура поиска пути к указанному файлу
run	Процедура выполнения вызываемой программы
mem_error1..4	Сообщения об ошибках памяти
child_error1..6	Сообщения об ошибках открываемого файла
terminate1..4	Сообщения о завершении работы

mem_allocated	Флаг очистки памяти (удалось или нет)
ss1, sp1	Слова для хранения стековых сегментов
psp	Слово для хранения адреса PSP
params	Блок параметров
cline	Параметры консоли
path	Массив для записи пути в будущем
child	Имя файла для запуска

Ход работы

1. Реализована процедура поиска пути к искомому файлу, получающая информацию из среды родительской программы.
2. Реализована процедура освобождения не нужной работающей программе памяти для выделения её вызываемой.
3. Реализована процедура выполнения и отслеживания результатов выполнения вызываемой программы.
4. Программа запущена и протестирована при условиях:
   a. После выполнения нажимается любая клавиша A-Z:

```
C:\>lab6
Память успешно очищена
Адрес недоступной памяти: 9FFF

Адрес среды: 01FE

Хвост консоли:

Содержимое среды:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Путь:
C:\LAB2.COM
a
Завершено с кодом a

C:\>_
```

b. После выполнения нажимается сочетание клавиш Ctrl+C:

```
C:\>lab6
Память успешно очищена
Адрес недоступной памяти: 9FFF

Адрес среды: 01FE

Хвост консоли:

Содержимое среды:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Путь:
C:\LAB2.COM
♥
Завершено с кодом ♥

C:\>
```

c. Программа запускается из другого каталога:

```
C:\>lab6\LAB6.EXE
Память успешно очищена
Адрес недоступной памяти: 9FFF

Адрес среды: 01FE

Хвост консоли:

Содержимое среды:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Путь:
C:\LAB6\LAB2.COM
a
Завершено с кодом a

C:\>
```

```
C:\>LAB6\LAB6.EXE
Память успешно очищена
Адрес недоступной памяти: 9FFF

Адрес среды: 01FE

Хвост консоли:

Содержимое среды:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Путь:
C:\LAB6\LAB2.COM
♥
Завершено с кодом ♥

C:\>
```

d. Модули находятся в разных каталогах:

```
C:\LAB6>LAB6.EXE
Память успешно очищена
Ошибка: Файл не найден

C:\LAB6>
```

Ответы на контрольные вопросы.

Сегментный адрес недоступной памяти

1. При нажатии Ctrl-Break управление передается по адресу (0000:008c). Адрес по вектору INT 23H копируется в поле PSP Ctrl-Break Address функциями DOS 26h и 4Ch. Нажатие Ctrl-Break вызывает немедленное прекращение работы программы.

2. В таком случае программа завершается после выполнения функции 4Ch прерывания int 21h.

3. Программа по прерыванию Ctrl-C заканчивается прямо в момент нажатия, когда срабатывает прерывание, в написанной программе это ожидание нажатия клавиши – функция 01h прерывания 21h.

Выводы.

В ходе выполнения лабораторной работы были связаны между собой два исполняемых модуля, один из которых вызывал другой. Изучены способы управления памятью и выполнением других приложений.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

- имя файла : lab6.asm

```
astack segment stack
      dw 64 dup(?)
astack ends

data segment
      mem_error1 db 'Ошибка: Разрушен управляющий блок памяти', 0dh, 0ah, '$'
      mem_error2 db 'Ошибка: Недостаточно памяти', 0dh, 0ah, '$'
      mem_error3 db 'Ошибка: Недействительный адрес памяти', 0dh, 0ah, '$'
      mem_error4 db 'Память успешно очищена' , 0dh, 0ah, '$'

      child_error1 db 'Ошибка: Недействительный номер функции', 0dh, 0ah, '$'
      child_error2 db 'Ошибка: Файл не найден', 0dh, 0ah, '$'
      child_error3 db 'Ошибка: сбой диска', 0dh, 0ah, '$'
      child_error4 db 'Ошибка: недостаточно памяти', 0dh, 0ah, '$'
      child_error5 db 'Ошибка: неправильная строка среды', 0dh, 0ah, '$'
      child_error6 db 'Ошибка: неверный формат', 0dh, 0ah, '$'

      terminate1 db 0dh, 0ah, 'Завершено с кодом    ' , 0dh, 0ah, '$'
      terminate2 db 0dh, 0ah, 'Завершено по Ctrl-Break' , 0dh, 0ah, '$'
      terminate3 db 0dh, 0ah, 'Завершено по ошибке устройства' , 0dh, 0ah,
'$'
      terminate4 db 0dh, 0ah, 'Завершено по функции 31h, программа оставлена
в памяти резидентно' , 0dh, 0ah, '$'

      mem_allocated db 0
      ss1 dw 0
      sp1 dw 0
      psp dw 0

      params dw 0
                  dd 0
                  dd 0
                  dd 0
      cline db 1, 0dh
      path db 64 dup(0)
      child db 'lab2.com', 0
data ends

code segment
assume cs:code, ds:data, ss:astack

free proc near
      push ax
      push bx
      push cx
      push dx

      mov ax, offset child + 9
      mov bx, offset codends
      add bx, ax
      mov cl, 4
      shr bx, cl
      add bx, 30h
```

```
	mov ah, 4ah
	int 21h
	jnc exit1

	mem_err1:
		cmp ax, 7
		jne mem_err2
		mov dx, offset mem_error1
		jmp mem_ret
	mem_err2:
		cmp ax, 8
		jne mem_err3
		mov dx, offset mem_error2
		jmp mem_ret
	mem_err3:
		mov dx, offset mem_error3
		jmp mem_ret
	exit1:
		mov mem_allocated, 1
		mov dx, offset mem_error4
	mem_ret:
		call print_string
		pop dx
		pop cx
		pop bx
		pop ax
		retn
free endp

find_path proc near
	push ax
	push bx
	push cx
	push dx
	push di
	push si
	push es

	mov ax, psp
	mov es, ax
	mov es, es:[2ch]
	mov bx, 0

	seek_path:
		inc bx
		cmp byte ptr es:[bx-1], 0
		jne seek_path
		cmp byte ptr es:[bx+1], 0
		jne seek_path

	add bx, 2
	mov di, 0

	write_dir:
		mov dl, es:[bx]
		mov byte ptr [path+di], dl
		inc di
		inc bx
		cmp dl, 0
		je quit
		cmp dl, '\'
```

```
            jne write_dir
            mov cx, di
            jmp write_dir
      quit:
            mov di, cx
            mov si, 0

      child_append:
            mov dl, byte ptr [child+si]
            mov byte ptr [path+di], dl
            inc di
            inc si
            cmp dl, 0
            jne child_append

      pop es
      pop si
      pop di
      pop dx
      pop cx
      pop bx
      pop ax
      retn
find_path endp

run proc near
      push ax
      push bx
      push cx
      push dx
      push ds
      push es
      mov sp1, sp
      mov ss1, ss

      mov ax, data
      mov es, ax
      mov bx, offset params
      mov dx, offset cline
      mov [bx+2], dx
      mov [bx+4], ds
      mov dx, offset path

      mov ax, 4b00h
      int 21h

      mov ss, ss1
      mov sp, sp1
      pop es
      pop ds

      jnc processing

      child_err1:
            cmp ax, 1
            jne child_err2
            mov dx, offset child_error1
            jmp exit2
      child_err2:
            cmp ax, 2
            jne child_err3
```

```
            mov dx, offset child_error2
            jmp exit2
      child_err3:
            cmp ax, 5
            jne child_err4
            mov dx, offset child_error3
            jmp exit2
      child_err4:
            cmp ax, 8
            jne child_err5
            mov dx, offset child_error4
            jmp exit2
      child_err5:
            cmp ax, 10
            jne child_err6
            mov dx, offset child_error5
            jmp exit2
      child_err6:
            mov dx, offset child_error6
            jmp exit2

      processing:
            mov ax, 4d00h
            int 21h

      termt1:
            cmp ah, 0
            jne termt2
            push di
            mov di, offset terminate1
            mov [di+20], al
            pop di
            mov dx, offset terminate1
            jmp exit2
      termt2:
            cmp ah, 1
            jne termt3
            mov dx, offset terminate2
            jmp exit2
      termt3:
            cmp ah, 2
            jne termt4
            mov dx, offset terminate3
            jmp exit2
      termt4:
            cmp ah, 3
            mov dx, offset terminate4

      exit2:
            call print_string
            pop dx
            pop cx
            pop bx
            pop ax
            retn
run endp

print_string proc near
      push ax
      mov ah, 9
      int 21h
```

```
      pop ax
      retn
print_string endp

main proc far
      push ds
      xor ax, ax
      push ax
      mov ax, data
      mov ds, ax
      mov psp, es

      call free
      cmp mem_allocated, 0
      je failed
      call find_path
      call run

      failed:
            xor al, al
            mov ah, 4ch
            int 21h
main endp
codends:
code ends
end main
```