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

Студентка гр. 0382 _____ Кривенцова Л.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2022

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.

Выполнение работы.

Функции, используемые в программе:

1. *PRINT* — вывод строки, адрес смещения до которой лежит в регистре DX.
2. *BYTE_TO_DEC* – процедура для перевода байт в число в десятичной системе счисления и записи в строку по адресу SI.
3. *free_memory* — процедура для высвобождение неиспользованной памяти.
4. make_optionsB — процедура, отвечающая за создание блока параметров.
5. *create_path*— процедура, создающая путь к вызываемому модулю.
6. *go*— процедура запуска вызываемого модуля.
7. *Main*—главная функция, вызывающая остальные действия.

Шаг 1.

На первом шаге был написан и отлажен .EXE модуль, который подготавливает параметры для запуска загрузочного модуля из той же

директории, в которой находится сам вызываемый модуль, а также запускает его, выводя сообщения о результатах работы и ошибках. В качестве вызываемого модуля была взята программа laba2, модифицированная таким образом, чтобы в конце она запрашивала у пользователя ввод символа с клавиатуры.

Шаг 2.

На втором шаге был запущен модуль .EXE и введен символ C.

Рис. 1 – результат запуска программы на втором шаге.

Видно, что при запуске модуля lab6.exe, память успешно была освобождена. Сразу после этого был вызван модуль laba2.com, где был введён символ. Распознавание символа работает верно, т.к. код ASCII символа C равен 67.

Шаг 3.

На третьем шаге был запущен модуль .EXE и введена комбинация Ctrl+C.

Рис. 2 - Результат работы программы на третьем шаге

Программа завершилась с соответствующим сообщением.

Шаг 4.

На четвертом шаге были произведены те же действия, как на втором и третьем шаге, с отличием в том, что теперь текущий каталог изменен (находится

не там же, где расположены файлы). Ниже представлены результаты запуска модуля .EXE при вводе символа C и комбинации Ctrl+C.

Рис. 3 - Результаты запуска программы на 4 шаге при вводе Cntrl-C.

Рис. 4 - Результаты запуска программы на 4 шаге при вводе символа «C».

Результаты идентичны с результатами на ранних шагах, потому что текущий каталог может быть любым (важно: при этом модули программ должны находиться в одной директории).

Шаг 5.

Произведён запуск программы, когда модули находятся в разных каталогах.

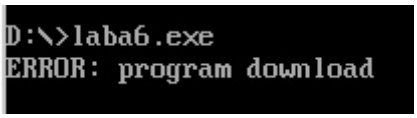

Рис. 5 – Результат запуска программы на пятом шаге.

Следовательно, когда загрузочные модули находятся в разных директориях, то запустить второй модуль с помощью первого невозможно.

Исходный код программы см. в приложении А.

Ответы на контрольные вопросы.

1. Как реализовано прерывание Ctrl+C?

- При нажатии сочетания клавиш Ctrl+C срабатывает прерывание int 23H и завершается работа текущей программы. Управление передается по адресу (адрес по вектору int 23h), который копируется в поле PSP (функциями 26H и 4CH). Потом исходное значение адреса восстанавливается из PSP при выходе из программы.

2. В какой точке заканчивается вызываемая программа, если код причины завершения 0?
   - Когда вызвана функция 4CH прерывания int 21H.
3. В какой точке заканчивается вызываемая программа по прерыванию Ctrl-C?
   - После функции 01H прерывания int 21H, когда ожидается ввод символа с клавиатуры.

Выводы.

Были исследованы возможности построения загрузочного модуля динамической структуры и интерфейс между вызывающим и вызываемым модулями по управлению и по данным.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: laba6.asm

```
ASTACK SEGMENT  STACK
DW 64 DUP(?)
ASTACK ENDS

DATA SEGMENT
OPTIONS    DB 14 dup(0)
PATH  DB 64 dup(0)
FILE  DB "laba2.COM", 0
ERROR1     DB 'ERROR: freeing memory!',13,10,'$'
ERROR2     DB 'ERROR: program download',13,10,'$'
ERROR3          DB 'Stop device error',13,10,'$'
EXIT       DB 13,10,'Completion is Okey.  Code:   67    ',13,10,'$'
C_EXIT          DB 'Ctrl-C            ',13,10,'$'
EXIT_31         DB 'Completion - function 31h ',13,10,'$'
KEEP_SS    DW ?
KEEP_SP    DW ?
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA, SS:ASTACK
BYTE_TO_DEC PROC NEAR
      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_l
      or AL,30h
      mov [SI],AL
      end_l: pop DX
      pop CX
      ret
BYTE_TO_DEC ENDP

PRINT PROC NEAR
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
PRINT ENDP

free_memory PROC NEAR
      push AX
```

```
	push BX
	push DX
	push CX

	mov BX, offset end_address
	mov AX, ES
	sub BX, AX
	mov CL, 4
	shr BX, CL
	mov AH, 4Ah
	int 21h

	jnc free_mem_end
	mov DX, offset ERROR1
	call PRINT

	free_mem_end:
	pop CX
	pop DX
	pop BX
	pop AX
	ret
free_memory ENDP

make_optionsB PROC NEAR
	push AX
	push DI
	mov DI, offset OPTIONS
	mov [DI+2], ES
	mov AX, 80h
	mov [DI+4], AX
	pop DI
	pop AX
	ret
make_optionsB ENDP

create_path PROC NEAR
	push DX
	push DI
	push SI
	push ES

	mov ES, ES:[2Ch]
	mov SI, offset PATH
	xor DI, DI

	read_byte:
	mov DL, ES:[DI]
	check_byte:
	inc DI
	cmp DL, 0
	jne read_byte
	mov DL, ES:[DI]
	cmp DL, 0
	jne check_byte

	add DI, 3
	write_path:
```

```
      mov DL, ES:[DI]
      mov [SI], DL
      inc SI
      inc DI
      cmp DL, 0
      jne write_path

      backslash_loop:
      mov DL, [SI-2]
      dec SI
      cmp DL, '\'
      jne backslash_loop

      mov DI, offset FILE
      write_filename:
      mov DL, [DI]
      mov [SI], DL
      inc SI
      inc DI
      cmp DL, 0
      jne write_filename

      pop ES
      pop SI
      pop DI
      pop DX
      ret
create_path ENDP

go PROC NEAR
      push AX
      push BX
      push DX
      push SI
      push ES

      mov KEEP_SS, SS
      mov KEEP_SP, SP
      mov AX, DS
      mov ES, AX
      mov BX, offset OPTIONS
      mov DX, offset PATH
      mov AX, 4B00h
      int 21h
      mov SS, KEEP_SS
      mov SP, KEEP_SP

      mov DX, offset ERROR2
      jc print_exit_info

      loaded:
      mov AH, 4Dh
      int 21h
      mov DX, offset EXIT
      cmp AH, 0
      je read_key
      mov DX, offset C_EXIT
      cmp AH, 1
```

```
            je print_exit_info
            mov DX, offset ERROR3
            cmp AH, 2
            je print_exit_info
            mov DX, offset ERROR3
            cmp AH, 3
            je print_exit_info

            read_key:
            mov SI, DX
            add SI, 28
            call BYTE_TO_DEC

            print_exit_info:
            call PRINT

            pop ES
            pop SI
            pop DX
            pop BX
            pop AX
            ret
      go ENDP

      Main PROC FAR
            sub AX, AX
            mov AX, DATA
            mov DS, AX
            call free_memory
            jc main_end
            call make_optionsB
            call create_path
            call go

            main_end:
            xor AL, AL
            mov AH, 4Ch
            int 21h
      Main ENDP

      end_address:
      CODE ENDS
end Main
```