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. 8382 _____ Наконечная А. Ю.

Преподаватель _____ Ефремов М. А.

Санкт-Петербург
2020

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры.

Постановка задачи.

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 передаётся 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 lab2, которая распечатывает среду и командную строку. Эту программу следует немного модифицировать, вставив перед выходом из неё обращение к функции ввода символа с клавиатуры. Введённое 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 отчёт.

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 отчёт.

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 отчёт.

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

Выполнение работы.

В работе исследуется интерфейс между вызывающим и вызываемым модулями по управлению и по данным. Для запуска вызываемого модуля используется функция 4B00h прерывания int 21h.

Были написаны функции, которые позволяют:

Подготовить место в памяти (FREE_MEMORY);

Создать блок параметров (PREPARE_BLOCK);

Подготовить строку, содержащую путь и имя вызываемой программы (FILE_NAME);

Вызвать загрузчик OS, при этом, если программа не была загружена, то устанавливается флаг переноса CF=1, а в AX заносится код ошибки (RUN).

Результат выполнения программы представлен на рисунках 1 — 4.

Рисунок 1 — Запуск программы, текущим каталогом является каталог с разработанными модулями, введён символ z

Рисунок 2 — Запуск программы, текущим каталогом является каталог с разработанными модулями, введена комбинация символов Ctrl-C

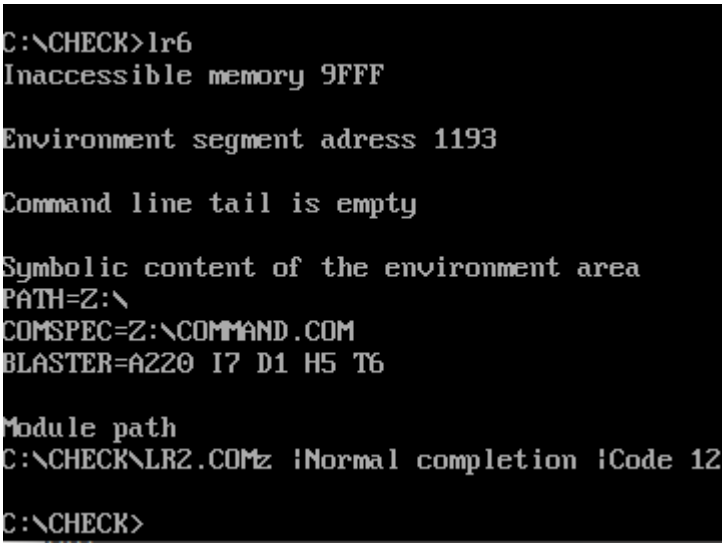

Рисунок 3 — Запуск программы, текущим каталогом является каталог отличный от того, в котором хранятся разработанные модули

Рисунок 4 — Запуск программы, модули находятся в разных каталогах

Ответы на контрольные вопросы.

1) Как реализовано прерывание Ctrl-C?

При обнаружении нажатия комбинации клавиш Ctrl-C выполняется команда int 23h. Это прерывание завершает текущий процесс и передаёт управление порождаемому процессу.

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

В точке вызова функции 4Ch прерывания int 21h.

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

В точке вызова функции 01h прерывания int 21h.

Выводы.

В ходе лабораторной работы была исследована возможность построения загрузочного модуля динамической структуры.

ПРИЛОЖЕНИЕ А

Исходный код программы

lr6.asm

```
ASTACK     SEGMENT  STACK
     dw 100 DUP(0)
ASTACK     ENDS

DATA  SEGMENT
     MEM_ERROR7_STR db 'Memory control block destroyed ',13,10,'$'
     MEM_ERROR8_STR db 'Not enough memory to execute function ',13,10,'$'
     MEM_ERROR9_STR db 'Invalid memory block address ',13,10,'$'
     PARAMETER_BLOCK dw 0 ;Сегментный адрес среды
                      dd 0 ;Сегмент и смещение командной строки
                      dd 0 ;Сегмент и смещение первого FCB
                      dd 0 ;сегмент и смещение второго FCB
     PATH_STR db 128 DUP (0)
     NAME_STR db 'lr2.com$'
     LOAD_ERROR1_STR db 'Function number is invalid ',13,10,'$'
     LOAD_ERROR2_STR db 'File not found ',13,10,'$'
     LOAD_ERROR5_STR db 'Disk error',13,10,'$'
     LOAD_ERROR8_STR db 'Not enough memory ',13,10,'$'
     LOAD_ERROR10_STR db 'Invalid environment string ',13,10,'$'
     LOAD_ERROR11_STR db 'Invalid format ',13,10,'$'
     KEEP_DS dw 0
     KEEP_SS dw 0
     KEEP_SP dw 0
     REASON0_STR     db ' |Normal completion |Code     ',13,10,'$'
     REASON1_STR db ' |Ctrl-Break Completion ',13,10,'$'
     REASON2_STR db ' |Device Error Termination ',13,10,'$'
     REASON3_STR db ' |Completion by function 31h ',13,10,'$'
DATA ENDS

CODE SEGMENT
   ASSUME CS:CODE,DS:DATA,SS:AStack

BYTE_TO_DEC PROC
; перевод в 10с/с, 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_l
      or AL, 30h
      mov [SI], AL
end_l:
      pop DX
      pop CX
      ret
BYTE_TO_DEC ENDP

RUN PROC
      push DS
      push ES
      mov KEEP_SP, SP
      mov KEEP_SS, SS
      mov AX, DS
      mov ES, AX
      mov DX, offset PATH_STR
      mov BX, offset PARAMETER_BLOCK
      mov AX, 4B00h
      int 21h
      mov SS, KEEP_SS
      mov sp, KEEP_SP
      pop ES
      pop DS
      push DX
      push AX
      push SI
      jc UNCORRECT
      mov AX, 4D00h
      int 21h
      mov DX, offset REASON1_STR
      cmp AH, 1
            je WRITE_REASON
      mov DX, offset REASON2_STR
      cmp AH, 2
            je WRITE_REASON
      mov DX, offset REASON3_STR
      cmp AH, 3
            je WRITE_REASON
      cmp AH, 0
            jne END_RUN
      mov DX, offset REASON0_STR
      mov SI, DX
      add SI, 28
      call BYTE_TO_DEC
```

```
      jmp WRITE_REASON
UNCORRECT:
      mov DX, offset LOAD_ERROR1_STR
      cmp AX, 1
            je WRITE_REASON
      mov DX, offset LOAD_ERROR2_STR
      cmp AX, 2
            je WRITE_REASON
      mov DX, offset LOAD_ERROR5_STR
      cmp AX, 5
            je WRITE_REASON
      mov DX, offset LOAD_ERROR8_STR
      cmp AX, 8
            je WRITE_REASON
      mov DX, offset LOAD_ERROR10_STR
      cmp AX, 10
            je WRITE_REASON
      mov DX, offset LOAD_ERROR11_STR
      cmp AX, 11
            je WRITE_REASON
WRITE_REASON:
      push AX
      mov AH, 09h
      int 21h
      pop AX
END_RUN:
      pop SI
      pop AX
      pop SI
      ret
RUN ENDP

FILE_NAME PROC
      push DX
      push DI
      push SI
      push ES
      xor DI, DI
      mov ES, ES:[2ch]
SKIP:
      mov DL, ES:[DI]
      cmp DL, 0
            je LAST
      inc DI
      jmp SKIP
LAST:
      inc DI
      mov DL, ES:[DI]
      cmp DL, 0
```

```
      jne SKIP
      add DI, 3
      mov SI, 0
PATH_WRITE:
      mov DL, ES:[DI]
      cmp DL, 0
            je DELETE
      mov PATH_STR[SI], DL
      inc DI
      inc SI
      jmp PATH_WRITE
DELETE:
      dec SI
      cmp PATH_STR[SI], 92
      je READY_FOR_ADDING
      jmp DELETE

READY_FOR_ADDING:
      mov di,-1
ADD_FILE_NAME:
      inc SI
      inc DI
      mov DL, NAME_STR[DI]
      cmp DL, '$'
      je SET_END
      mov PATH_STR[SI], DL
      jmp ADD_FILE_NAME
SET_END:
      pop ES
      pop SI
      pop DI
      pop DX
      ret
FILE_NAME ENDP

PREPARE_BLOCK PROC
      push AX
      push BX
      push DX
      mov BX, offset PARAMETER_BLOCK
      mov DX, ES
      mov AX, 0
      mov [BX], AX
      mov [BX + 2], DX
      mov AX, 80h
      mov [BX + 4], AX
      mov [BX + 6], DX
      mov  AX, 5Ch
      mov [BX + 8], AX
```

```
      mov [BX + 10], DX
      mov AX, 6Ch
      mov [BX + 12], AX
      pop  DX
      pop BX
      pop AX
      ret
PREPARE_BLOCK ENDP

FREE_MEMORY PROC
      push AX
      push BX
      mov BX, 4096
      mov AH, 4Ah
      int 21h
      jnc MEM_END
      mov DX, offset MEM_ERROR7_STR
      cmp AX,7
           je MEM_WRITE
      mov DX, offset MEM_ERROR8_STR
      cmp AX,8
           je MEM_WRITE
      mov DX, offset MEM_ERROR9_STR
      cmp AX,9
           je MEM_WRITE
MEM_END:
      pop BX
      pop AX
      ret
MEM_WRITE:
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
FREE_MEMORY ENDP

Main PROC FAR
      mov AX, DATA
      mov DS, AX
      call FREE_MEMORY
      call PREPARE_BLOCK
      call FILE_NAME
      call RUN

    ;Выход в DOS
      xor AL,AL
      mov AH,4Ch
      int 21h
```

```
Main ENDP
ENDING:
      CODE ENDS
      END Main
```