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. 8382 _____ Колногоров Д.Г.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры.

Выполнение работы.

Был написан программный модуль типа **.EXE** (представлен в приложении А), который выполяет следующие 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 модулль запускается с использованием загрузчика.

3) После запуска проверяется выполнение загрузчика, а затем результат выполения вызываемой программы. Проверяется причина завершения и, в зависимости от значения, выводится соответствующее сообщение. Если причина завершения 0, то выводится код завершения.

В качестве вызываемой программы была взята программа ЛР 2 (код представлен в приложении Б), которая распечатывает среду и командную строку. Программа была модифицирована таким образом, чтобы перед выходом вызывалась функция ввода символа с клавиатуры.

На рисунке 1 представлен результат работы программы в случае, когда для завершения работы вызываемой программы была нажата клавиша q.

Рисунок 1 — результат работы программы при нажатии q

На рисунке 2 представлен результат работы программы в случае, когда для завершения работы вызываемой программы была нажата комбинация клавиш Ctrl-C.

Рисунок 2 — результат работы программы при нажатии Ctrl-C

На рисунках 3 и 4 представлены результаты работы программы при запуске из каталога *TEST*, и нажатии для выхода клавиш q и Ctrl-C соответственно.

Рисунок 3 — результат работы программы при запуске из каталога TEST и нажатии q

Рисунок 4 — результат работы программы при запуске из каталога TEST и нажатии Ctrl-C

Результат работы программы при отсутствии вызываемого программой модуля в каталоге программы предствлен на рисунке 5.

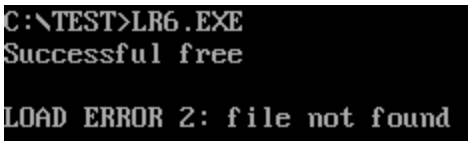

Рисунок 5 — результат работы программы при отсутствии вызываемого модуля

Контрольные вопросы.

1) Как реализовано прерывание Ctrl-C?

Когда пользователь нажимает Ctrl-C (или Ctrl-Break) в буфер клавиатуры помещается соответствующий код клавиш. Когда DOS обнаруживает наличие в буфере данного кода вызывается инструкция 23h. Затем обработчик, установленный DOSом, выполняет все необходимые операции по завершению работы программы.

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

Код причины завершения 0 говорит о нормальном завершении программы, то есть программы завершилась в точке вызова функции 4Ch прерывания 21h (функция завершения программы).

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

При прерывании Ctrl-C управление передаётся инструкции 23h, которая завершает работу программы.

Вывод.

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

ПРИЛОЖЕНИЕ А

СОДЕРЖИМОЕ ФАЙЛА LR26.ASM

```
DATA SEGMENT
      NEW_LINE db 10,13,'$'

      PROGRAM_TO_RUN           db "LR2.COM",0
      PROGRAM_TO_RUN_FULL_PATH db 128 dup(0)

      COMMAND_LINE db 1h, 0Dh      ; first byte is size of cmd, 0Dh=carriage
return

      STR_MEMORY_SUCCESS db "Successful free",10,13,"$"
      STR_MEMORY_ERROR7  db "FREE ERROR 7: controlling block is
destroyed",10,13,"$"
      STR_MEMORY_ERROR8  db "FREE ERROR 8: not enough memory for
function",10,13,"$"
      STR_MEMORY_ERROR9  db "FREE ERROR 9: wrong memory block address",10,13,"$"

      STR_LOAD_SUCCESS db "Successful load",10,13,"$"
      STR_LOAD_ERROR1  db "LOAD ERROR 1: wrong function number",10,13,"$"
      STR_LOAD_ERROR2  db "LOAD ERROR 2: file not found",10,13,"$"
      STR_LOAD_ERROR5  db "LOAD ERROR 5: disk error",10,13,"$"
      STR_LOAD_ERROR8  db "LOAD ERROR 8: not enough memory",10,13,"$"
      STR_LOAD_ERROR10 db "LOAD ERROR 10: wrong env string",10,13,"$"
      STR_LOAD_ERROR11 db "LOAD ERROR 11: ",10,13,"$"

      STR_EXIT_REASON0 db "Exit reason: normal exit with code  ",10,13,"$"
      STR_EXIT_REASON1 db "Exit reason: Ctrl-Break",10,13,"$"
      STR_EXIT_REASON2 db "Exit reason: device error",10,13,"$"
      STR_EXIT_REASON3 db "Exit reason: int 31h (resident)",10,13,"$"

      PARAMETERS_BLOCK dw 0 ; seg address of env
                            dd 0 ; seg and offset of cmd
                            dd 0
                            dd 0

      KEEP_SS  dw 0
      KEEP_SP  dw 0
      KEEP_PSP dw 0
```

```
      END_OF_DATA db 0

DATA ENDS

AStack SEGMENT STACK
      DW 200 DUP(?)
AStack ENDS

CODE SEGMENT

      ASSUME  CS:CODE, DS:DATA, ES:NOTHING, SS:AStack

      PRINT_NEW_LINE PROC NEAR
            push DX
            push AX

            mov DX, offset NEW_LINE
            mov AH, 09h
            int 21h

            pop AX
            pop DX
            ret
      PRINT_NEW_LINE ENDP

      PRINT_STRING PROC NEAR
            push DX
            push AX

            mov AH, 09h
            int 21h

            pop AX
            pop DX
            ret
      PRINT_STRING ENDP

      PREPARE_MEMORY PROC NEAR
            ; AX=1 if successful free
            ; otherwise AX=0
            push BX
```

```
push DX

mov BX, offset END_OF_PROGRAM
mov AX, offset END_OF_DATA
add BX, AX

push CX
mov CL, 4
shr BX, CL
add BX, 2Bh
pop CX

mov AH, 4Ah
int 21h

jnc MEMORY_SUCCESS

cmp AX, 7
je MEMORY_ERROR7
cmp AX, 8
je MEMORY_ERROR7
cmp AX, 9
je MEMORY_ERROR7

MEMORY_ERROR7:
      mov DX, offset STR_MEMORY_ERROR7
      jmp MEMORY_FAIL
MEMORY_ERROR8:
      mov DX, offset STR_MEMORY_ERROR8
      jmp MEMORY_FAIL
MEMORY_ERROR9:
      mov DX, offset STR_MEMORY_ERROR9
      jmp MEMORY_FAIL

MEMORY_SUCCESS:
      mov AX, 1
      mov DX, offset STR_MEMORY_SUCCESS
      call PRINT_STRING
      jmp PREPARE_MEMORY_END

MEMORY_FAIL:
```

```
            mov AX, 0
            call PRINT_STRING

PREPARE_MEMORY_END:
      pop DX
      pop BX

      ret
PREPARE_MEMORY ENDP

PREPARE_PATH PROC NEAR
      push AX
      push CX
      push BX
      push DI
      push SI
      push ES

      ; set ES to env variables segment
      mov AX, KEEP_PSP
      mov ES, AX
      mov ES, ES:[2Ch]

      mov BX, 0
      print_env_variable:
            cmp BYTE PTR ES:[BX], 0
            je variable_end
            inc BX
            jmp print_env_variable
      variable_end:
            inc BX
            cmp BYTE PTR ES:[BX+1], 0
            jne print_env_variable

      add BX, 2   ; skip 0 and space

      mov DI, 0
      path_loop:
            mov DL, ES:[BX]
            mov BYTE PTR [PROGRAM_TO_RUN_FULL_PATH+DI], DL
            inc BX
```

```
            inc DI
            cmp DL, 0
            je path_loop_end
            cmp DL, '\'
            jne path_loop
            mov CX, DI
            jmp path_loop
      path_loop_end:
      mov DI, CX

      mov SI, 0
      filename_loop:
            mov DL, BYTE PTR [PROGRAM_TO_RUN+SI]
            mov BYTE PTR [PROGRAM_TO_RUN_FULL_PATH+DI], DL
            inc DI
            inc SI
            cmp DL, 0
            jne filename_loop

      pop ES
      pop SI
      pop DI
      pop BX
      pop CX
      pop AX

      ret
PREPARE_PATH ENDP

LOAD_PROGRAM PROC NEAR
      push AX
      push BX
      push CX
      push DX

      push DS
      push ES
      mov KEEP_SP, SP
      mov KEEP_SS, SS

      mov AX, DATA
```

```
mov ES, AX
mov BX, offset PARAMETERS_BLOCK
mov DX, offset COMMAND_LINE
mov [BX+2], DX          ; set offset of cmd
mov [BX+4], DS          ; set seg addr of cmd
mov DX, offset PROGRAM_TO_RUN_FULL_PATH

mov AX, 4B00h
int 21h

mov SS, KEEP_SS
mov SP, KEEP_SP
pop ES
pop DS

jnc LOAD_SUCCESS

cmp AX, 1
je LOAD_ERROR1
cmp AX, 2
je LOAD_ERROR2
cmp AX, 5
je LOAD_ERROR5
cmp AX, 8
je LOAD_ERROR8
cmp AX, 10
je LOAD_ERROR10
cmp AX, 11
je LOAD_ERROR11

LOAD_ERROR1:
      mov DX, offset STR_LOAD_ERROR1
      jmp LOAD_FAIL
LOAD_ERROR2:
      mov DX, offset STR_LOAD_ERROR2
      jmp LOAD_FAIL
LOAD_ERROR5:
      mov DX, offset STR_LOAD_ERROR5
      jmp LOAD_FAIL
LOAD_ERROR8:
      mov DX, offset STR_LOAD_ERROR8
```

```
      jmp LOAD_FAIL
LOAD_ERROR10:
      mov DX, offset STR_LOAD_ERROR10
      jmp LOAD_FAIL
LOAD_ERROR11:
      mov DX, offset STR_LOAD_ERROR11
      jmp LOAD_FAIL

LOAD_SUCCESS:
      call PRINT_NEW_LINE
      mov DX, offset STR_LOAD_SUCCESS
      call PRINT_STRING

      ; get exit code
      mov AH, 4Dh
      mov AL, 00h
      int 21h

      cmp AH, 0
      je EXIT_REASON0
      cmp AH, 1
      je EXIT_REASON1
      cmp AH, 2
      je EXIT_REASON2
      cmp AH, 3
      je EXIT_REASON3

      EXIT_REASON0:
            ; write exit button to string
            push DI
            mov DI, offset STR_EXIT_REASON0
            mov [DI+35], AL
            pop DI
            mov DX, offset STR_EXIT_REASON0
            jmp LOAD_SUCCESS_END
      EXIT_REASON1:
            mov DX, offset STR_EXIT_REASON1
            jmp LOAD_SUCCESS_END
      EXIT_REASON2:
            mov DX, offset STR_EXIT_REASON2
            jmp LOAD_SUCCESS_END
```

```
            EXIT_REASON3:
                  mov DX, offset STR_EXIT_REASON3
                  jmp LOAD_SUCCESS_END

            LOAD_SUCCESS_END:
            call PRINT_STRING

            jmp LOAD_END

      LOAD_FAIL:
            call PRINT_NEW_LINE
            call PRINT_STRING

      LOAD_END:

      pop DX
      pop CX
      pop BX
      pop AX

      ret
LOAD_PROGRAM ENDP

MAIN PROC
      PUSH DS
      SUB AX, AX
      PUSH AX
      MOV AX, DATA
      MOV DS, AX
      mov KEEP_PSP, ES

      call PREPARE_MEMORY
      cmp AX, 1
      jne MAIN_END

      call PREPARE_PATH

      call LOAD_PROGRAM

      MAIN_END:
      xor AL, AL
```

```
            mov AH, 4Ch
            int 21h
      MAIN ENDP
      END_OF_PROGRAM:

CODE ENDS

END MAIN
```

ПРИЛОЖЕНИЕ Б

СОДЕРЖИМОЕ ФАЙЛА LR2.ASM

```
TESTPC      SEGMENT
            ASSUME  CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
            ORG     100H

START:      JMP     BEGIN
; DATA
NEW_LINE           DB 10,13,'$'
STR_SEG_MEM        DB 'UNAVAILABLE MEMORY: $'
STR_ENV_ADDR       DB 'ENVIRONMENT MEMORY: $'
STR_TAIL           DB 'TAIL: $'
STR_NO_TAIL        DB 'NO TAIL',10,13,'$'
STR_ENV_VARIABLES DB 'ENVIRONMENT VARIABLES:',10,13,'$'
STR_PATH           DB 'PATH: $'

PRINT_NEW_LINE  PROC NEAR
      PUSH DX
      PUSH AX

      MOV DX, OFFSET NEW_LINE
      MOV AH, 09H
      INT 21H

      POP AX
      POP DX
      RET
PRINT_NEW_LINE  ENDP

PRINT_BYTE      PROC NEAR
; PRINTS AL AS TWO HEX DIGITS
      PUSH BX
      PUSH DX

      CALL BYTE_TO_HEX
      MOV BH, AH

      MOV DL, AL
      MOV AH, 02H
      INT 21H
```

```
      MOV DL, BH
      MOV AH, 02H
      INT 21H

      POP DX
      POP BX
      RET
PRINT_BYTE    ENDP
TETR_TO_HEX      PROC NEAR
      AND      AL,0FH
      CMP      AL,09
      JBE      NEXT
      ADD      AL,07
NEXT:
      ADD      AL,30H
      RET
TETR_TO_HEX   ENDP
;--------------------------------
BYTE_TO_HEX   PROC  NEAR
; AL --> TWO HEX SYMBOLS IN AX
      PUSH     CX
      MOV      AH,AL
      CALL     TETR_TO_HEX
      XCHG     AL,AH
      MOV      CL,4
      SHR      AL,CL
      CALL     TETR_TO_HEX ; AL - HIGH DIGIT
      POP      CX          ; AH - LOW DIGIT
      RET
BYTE_TO_HEX  ENDP
;--------------------------------
; CODE
BEGIN:

PRINT_SEG_MEM:
      MOV DX, OFFSET STR_SEG_MEM
      MOV AH, 09H
      INT 21H

      MOV BX, DS:[02H]
```

```
      MOV AL, BH
      CALL PRINT_BYTE
      MOV AL, BL
      CALL PRINT_BYTE

      CALL PRINT_NEW_LINE
PRINT_ENV_ADDR:
      MOV DX, OFFSET STR_ENV_ADDR
      MOV AH, 09H
      INT 21H

      MOV BX, DS:[2CH]
      MOV AL, BH
      CALL PRINT_BYTE
      MOV AL, BL
      CALL PRINT_BYTE

      CALL PRINT_NEW_LINE
PRINT_TAIL:
      MOV DX, OFFSET STR_TAIL
      MOV AH, 09H
      INT 21H

      MOV CH, 0
      MOV CL, DS:[80H]
      CMP CL, 0
      JE NO_TAIL

      MOV BX, 0
      TAIL_LOOP:
            MOV DL, DS:[81H+BX]
            MOV AH, 02H
            INT 21H

            INC BX
            LOOP TAIL_LOOP

      CALL PRINT_NEW_LINE
      JMP TAIL_END
      NO_TAIL:
      MOV DX, OFFSET STR_NO_TAIL
```

```
	MOV AH, 09H
	INT 21H
	TAIL_END:

PRINT_ENV_CONTENTS:
	MOV DX, OFFSET STR_ENV_VARIABLES
	MOV AH, 09H
	INT 21H

	MOV ES, DS:[2CH]
	MOV BX, 0
	PRINT_ENV_VARIABLE:
		MOV DL, ES:[BX]
		CMP DL, 0
		JE NEW_VARIABLE

		MOV AH, 02H
		INT 21H

		JMP VARIABLE_END
	NEW_VARIABLE:
		CALL PRINT_NEW_LINE
	VARIABLE_END:
		INC BX
		MOV DX, ES:[BX]
		CMP DX, 0
		JE ENV_CONTENT_END
		JMP PRINT_ENV_VARIABLE
	ENV_CONTENT_END:
	CALL PRINT_NEW_LINE

	ADD BX, 1

PRINT_MODULE_PATH:
	MOV DX, OFFSET STR_PATH
	MOV AH, 09H
	INT 21H
	ADD BX, 3
	PATH_LOOP:
		MOV DL, ES:[BX]
		CMP DL, 0
```

```
            JNE PATH_NEXT
            JE LOOP_END
      PATH_NEXT:
            MOV AH, 02H
            INT 21H
            INC BX
            JMP PATH_LOOP
      LOOP_END:
      CALL PRINT_NEW_LINE

      MOV AH, 01H
      INT 21H

; RETURN TO DOS
            MOV     AH,4CH
            INT     21H
TESTPC      ENDS
            END     START     ; MODULE END START - ENTRY POINT
```