**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Операционные системы»**

**Тема: Исследование организации управления основной пямятью**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 |  | Мололкин К.А. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2020

**Цель работы**

Исследовать структуры данных и работу функций управления памятью ядра операционной системы.

**Ход работы**

Был написан и отлажен программный модуль .COM, который выбирает и распечатывает количество доступной памяти, размер расширенной памяти, цепочку блоков управления памятью. Результат работы программы представлен на рис. 1.

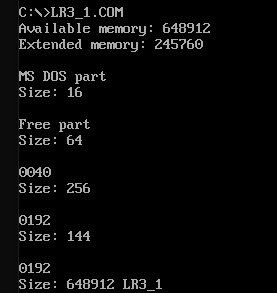


Рисунок 1 – работа программы после первого шага

На следующем шаге программа была изменена таким образом, чтобы она освобождала память, которую она не занимает, с помощью функции 4Аh прерывания 21h. Результат работы программы представлен на рис. 2.

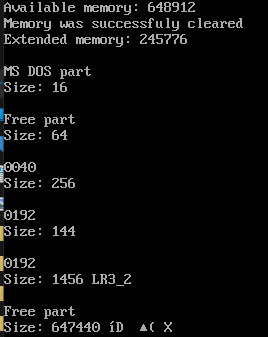
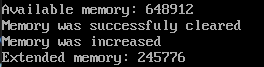


Рисунок 2 – работа программы после второго шага

На третьем шаге программа была изменена еще раз таким образом, чтобы после освобождения памяти программа запрашивала 64Кб памяти функцией 48Н прерывания 21Н. Результат работы программы представлен на рис. 3.



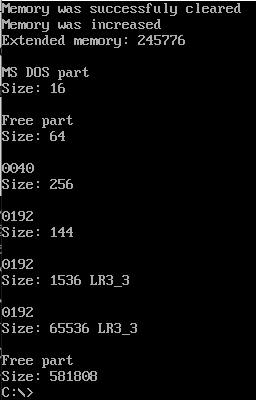


Рисунок 3 – работа программы после третьего шага

На четвертом шаге программа была изменена таким образом, что запрос памяти производится до ее освобождения. Результат работы программы представлен на рис. 4. Код программы представлен в приложении А.

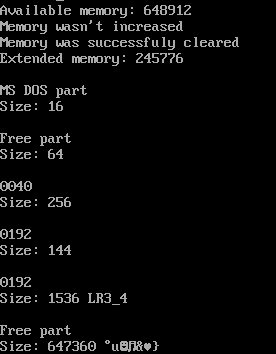


Рисунок 4 – работа программы после четвертого шага

Контрольные вопросы

1. Что означает «доступный объем памяти»?

Доступный объём памяти – это память, доступная программам для их выполнения.

1. Где MCB блок вашей программы в списке?

Для программ написанных на 1, 2 и 4 шагах это последний блок, а для программы, написанной на 3 шаге это последний и предпоследний блоки.

1. Какой размер памяти занимает программа в каждом случае?

В первом случае 648912 байт – вся доступная память.

Во втором случае 1456 байт – столько было запрошено при освобождении.

В третьем случае (1536 + 65535) байт, так как память сначала была очищена, а потом добавилась при выделении.

В четвертом случае 1536, так как выделения не было.

Вывод

Во время выполнения лабораторной работы были исследованы структуры данных и работа функций управления памятью ядра операционной системы.

Приложение А.

(Код программы)

LR3 SEGMENT

ASSUME CS:LR3, DS:LR3, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

;DATA

AVAILABLE\_MEMORY db "Available memory: $"

EXTENDED\_MEMORY db "Extended memory: $"

ENTER db 13, 10, "$"

SPACE db " $"

FREE\_PART\_MEM db 13, 10, "Free part$"

OS\_XMSUMP\_PART db 13, 10, "OS XMS UMB driver part$"

UPPER\_MEM\_PART db 13, 10, "Excluded upper driver memory part$"

MS\_DOS\_PART db 13, 10, "MS DOS part$"

CU\_386MAX\_PART db 13, 10, "Control unit 386 MAX UMB occupied part$"

BLOCKED\_386MAX\_PART db 13, 10, "386 MAX blocked part$"

PART\_386MAX\_UMD db 13, 10, "386 MAX UMB part$"

ANOTHER\_OWNER db 13, 10, " $"

MEM\_SIZE db 13, 10, "Size: $"

SUCCESS\_CLEAR db "Memory was successfuly cleared", 13, 10,"$"

FAILD\_CLEAR db "Memory wasn't cleared",13, 10,"$"

MEM\_INCREASE\_SECC db "Memory was increased",13, 10,"$"

MEM\_INCREASE\_FAIL db "Memory wasn't increased",13, 10,"$"

;PROCEDURES

WRD\_TO\_HEX PROC near

push BX

mov BH,AH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

dec DI

mov AL,BH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

pop BX

ret

WRD\_TO\_HEX 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

push CX

mov AH,AL

call TETR\_TO\_HEX

xchg AL,AH

mov CL,4

shr AL,CL

call TETR\_TO\_HEX

pop CX

ret

BYTE\_TO\_HEX ENDP

PRINT\_STRING PROC near

push AX

mov ah, 09h

int 21h

pop AX

ret

PRINT\_STRING ENDP

PRINT\_NUM Proc near

mov bx,0ah

xor cx,cx

del:

div bx

push dx

inc cx

xor dx,dx

cmp ax,0

jnz del

print\_symb:

pop dx

or dl,30h

call PRINT\_SYMBOL

loop print\_symb

ret

PRINT\_NUM ENDP

PRINT\_SYMBOL Proc near

push ax

mov ah, 02h

int 21h

pop ax

ret

PRINT\_SYMBOL ENDP

PRINT\_AVAILABLE\_MEMORY PROC near

mov dx, offset AVAILABLE\_MEMORY

call PRINT\_STRING

mov ah, 4Ah

mov bx, 0FFFFh

int 21h

mov ax, bx

mov bx, 10h

mul bx

call PRINT\_NUM

mov dx, offset ENTER

call PRINT\_STRING

ret

PRINT\_AVAILABLE\_MEMORY ENDP

PRINT\_EXTENDED\_MEMORY PROC near

mov dx, offset EXTENDED\_MEMORY

call PRINT\_STRING

mov al, 30h

out 70h, al

in al, 71h

mov bl, al

mov al,31h

out 70h, al

in al, 71h

mov bh, ah

mov ah, al

mov al, bh

mov bx,10h

mul bx

call PRINT\_NUM

ret

PRINT\_EXTENDED\_MEMORY ENDP

PRINT\_MCB PROC near

mov ah, 52h

int 21h

mov ax, es:[bx-2]

mov es, ax

xor cx, cx

print\_next:

inc cx

mov dx, offset ENTER

call PRINT\_STRING

push cx

xor ax, ax

mov al, es:[0h]

push ax

mov ax, es:[1h]

cmp ax, 0h

je free\_part

cmp ax, 6h

je driver\_part

cmp ax, 7h

je upper\_memory\_part

cmp ax, 8h

je msdos\_part

cmp ax, 0FFFAh

je control\_386max\_umb

cmp ax, 0FFFDh

je blocked\_386max

cmp ax, 0FFFEh

je p\_386max\_umd

xor dx, dx

mov di, offset ANOTHER\_OWNER

add di, 5

call WRD\_TO\_HEX

mov dx, offset ANOTHER\_OWNER

jmp end\_c\_part

free\_part:

mov dx, offset FREE\_PART\_MEM

jmp end\_c\_part

driver\_part:

mov dx, offset OS\_XMSUMP\_PART

jmp end\_c\_part

upper\_memory\_part:

mov dx, offset UPPER\_MEM\_PART

jmp end\_c\_part

msdos\_part:

mov dx, offset MS\_DOS\_PART

jmp end\_c\_part

control\_386max\_umb:

mov dx, offset CU\_386MAX\_PART

jmp end\_c\_part

blocked\_386max:

mov dx, offset BLOCKED\_386MAX\_PART

jmp end\_c\_part

p\_386max\_umd:

mov dx, offset PART\_386MAX\_UMD

end\_c\_part:

call PRINT\_STRING

mov dx, offset MEM\_SIZE

call PRINT\_STRING

mov ax, es:[3h]

mov bx, 10h

mul bx

call PRINT\_NUM

mov dx, offset SPACE

call PRINT\_STRING

mov cx, 8

xor si, si

end\_mem:

mov dl, es:[si+8h]

mov ah, 02h

int 21h

inc si

loop end\_mem

mov ax, es:[3h]

mov bx, es

add bx, ax

inc bx

mov es, bx

pop ax

pop cx

cmp al, 5Ah

je p\_end

jmp print\_next

p\_end:

ret

PRINT\_MCB ENDP

CLEAR\_MEM Proc near

mov ah, 4Ah

mov bx, offset END\_STACK

add bx, 10Fh

shr bx, 1

shr bx, 1

shr bx, 1

shr bx, 1

int 21h

jnc good\_clear

mov dx, offset FAILD\_CLEAR

jmp pp\_end

good\_clear:

mov dx, offset SUCCESS\_CLEAR

pp\_end:

call PRINT\_STRING

ret

CLEAR\_MEM ENDP

INCREASE\_MEM Proc near

mov bx, 1000h

mov ah, 48h

int 21h

jnc success\_inc

mov dx, offset MEM\_INCREASE\_FAIL

jmp end\_p

success\_inc:

mov dx, offset MEM\_INCREASE\_SECC

end\_p:

call PRINT\_STRING

ret

INCREASE\_MEM ENDP

BEGIN:

call PRINT\_AVAILABLE\_MEMORY

call INCREASE\_MEM

call CLEAR\_MEM

call PRINT\_EXTENDED\_MEMORY

call PRINT\_MCB

xor AL, AL

mov AH, 4Ch

int 21h

STACK\_DEC:

DW 128 dup (0)

END\_STACK:

LR3 ENDS

END START