

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 памятью

Студент гр. 8383

Мололкин К.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

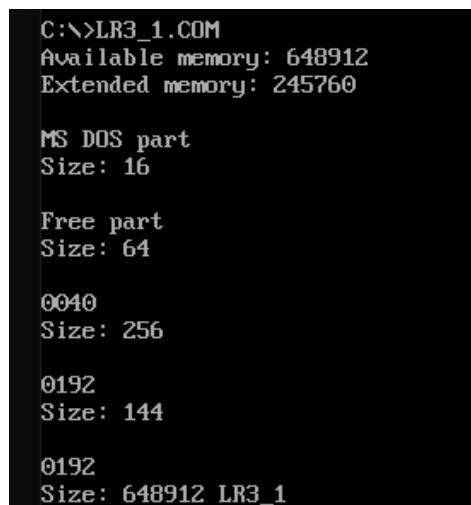
2020

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.



```
C:\>LR3_1.COM
Available memory: 648912
Extended memory: 245760

MS DOS part
Size: 16

Free part
Size: 64

0040
Size: 256

0192
Size: 144

0192
Size: 648912 LR3_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 4Ah прерывания 21h. Результат работы программы представлен на рис. 2.

```

Available memory: 648912
Memory was successfully cleared
Extended memory: 245776

MS DOS part
Size: 16

Free part
Size: 64

0040
Size: 256

0192
Size: 144

0192
Size: 1456 LR3_2

Free part
Size: 647440 ID  ▲( X

```

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 48H прерывания 21H. Результат работы программы представлен на рис. 3.

```

Available memory: 648912
Memory was successfully cleared
Memory was increased
Extended memory: 245776

MS DOS part
Size: 16

Free part
Size: 64

0040
Size: 256

0192
Size: 144

0192
Size: 1536 LR3_3

0192
Size: 65536 LR3_3

Free part
Size: 581808
C:\>

```

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.

```
Available memory: 648912
Memory wasn't increased
Memory was successfully cleared
Extended memory: 245776

MS DOS part
Size: 16

Free part
Size: 64

0040
Size: 256

0192
Size: 144

0192
Size: 1536 LR3_4

Free part
Size: 647360 °u0026♥}
```

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.

2. Где МСВ блок вашей программы в списке?

Для программ написанных на 1, 2 и 4 шагах это 4 и 5 блоки, а для программы, написанной на 3 шаге то 4, 5, 6.

3. Какой размер памяти занимает программа в каждом случае?

В первом случае $648912 + 144 = 649056$ байт – вся доступная память.

Во втором случае $144 + 1456 + 647440 = 649040$ байт – столько было запрошено при освобождении.

В третьем случае $144 + 1536 + 65535 = 67216$ байт, так как память сначала была очищена, а потом добавилась при выделении.

В четвертом случае $144 + 1536 = 1680$ байт, 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: $"
SUCCESS_CLEAR db "Memory was successfully cleared", 13,
10, "$"
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

```