ЛР1

int_8h_listing

```
1.                  ; -- Вызов sub_04
2.                  call    sub_4                   ; (07B9)
3.
4.                  ; -- Сохранение регистров es, ds, ax, dx
5.                  push    es
6.                  push    ds
7.                  push    ax
8.                  push    dx
9.
10.                 ; -- Загрузка в DS 0040h
11.                 mov     ax,40h
12.                 mov     ds,ax
13.
14.                 ; -- AX = ES = 0
15.                 xor     ax,ax                   ; Zero register
16.                 mov     es,ax
17.
18.                 ; -- Инкремент счетчика таймера по адресу 0040:006C
19.                 inc     word ptr ds:[6Ch]       ; (0040:006C=22A7h)
20.                 jnz     loc_3                   ; Jump if not zero
21.
22.                 ; -- Инкремент старших двух байта счетчика таймера
23.                 inc     word ptr ds:[6Eh]       ; (0040:006E=16h)
24.
25. ; -- Проверка на то, что прошло 24 часа
26. ; 0040H:006EH == 18H и 0040H:006H = 00B0H
27. ; 24 * 60 * 60 * t == 18H << 16 + B0H, где количество вызовов таймера в секунду - это t
28. loc_3:
29.                 cmp     word ptr ds:[6Eh],18h ; (0040:006E=16h)
30.                 jne     loc_4                   ; Jump if not equal
31.                 cmp     word ptr ds:[6Ch],0B0h        ; (0040:006C=22A7h)
32.                 jne     loc_4                   ; Jump if not equal
33.
34.                 ; -- Зануление счетчика таймера и занесение 1 в 0040H:0070 тогда, когда прошло
    24 часа
35.                 mov     word ptr ds:[6Eh],ax  ; (0040:006E=16h)
36.                 mov     word ptr ds:[6Ch],ax  ; (0040:006C=22A7h)
37.                 mov     byte ptr ds:[70h],1   ; (0040:0070=0)
38.
39.                 ; -- Ранее AL = 0, теперь AL = 8
40.                 or      al,8
41. loc_4:
42.                 ; -- Сохранение регистра AX
43.                 push    ax
44.
45.                 ; -- Декрементирование счетчика отключения моторчика
46.                 dec     byte ptr ds:[40h]       ; (0040:0040=35h)
47.                 jnz     loc_5                   ; Jump if not zero
48.
49.                 ; -- Установка флагов, отвечающих за отключение моторчика дисковода
50.                 and     byte ptr ds:[3Fh],0F0h        ; (0040:003F=0)
51.                 mov     al,0Ch
52.                 mov     dx,3F2h
53.                 out     dx,al                   ; port 3F2h, dsk0 contrl output
54. loc_5:
55.                 ; -- Восстановление регистра AX
56.                 pop     ax
57.
58.                 ; -- Проверка 2 бита, Parity Flag
59.                 test    word ptr ds:[314h],4  ; (0040:0314=3200h)
60.                 jnz     loc_6                   ; Jump if not zero
61.
62.                 ; -- Загрузка младшего байта FLAGS в регистр AH
63.                 lahf                            ; Load ah from flags
64.
65.                 xchg    ah,al
66.                 push    ax
67.
68.                 ; -- Выхов 1CH с помощью адреса в таблице векторов. При вызове call на месте
    регистра будет лежать AX, который по выходу из 1CH будет установлен в FLAGS с помощью IRET
69.                 call    dword ptr es:[70h]    ; (0000:0070=6ADh)
70.                 jmp     short loc_7           ; (07A5)
```

```
71.             nop
72. loc_6:
73.             int     1Ch                     ; Timer break (call each 18.2ms)
74. loc_7:
75.             call    sub_4                   ; (07B9)
76.
77.             ; -- Сброс котроллера прерываний
78.             mov     al,20h                  ; ' '
79.             out     20h,al                  ; port 20h, 8259-1 int command
80.                                             ;  al = 20h, end of interrupt
81.
82.             ; -- Восстановление регистров dx, ax, ds, es
83.             pop     dx
84.             pop     ax
85.             pop     ds
86.             pop     es
87.
88.             jmp     $-164h ; (020F:07B0H - 164h = 020A:064Ch)
89.
90.             db      0C4h
91.
92.             les     cx,dword ptr ds:[93E9h]       ; (0000:93E9=0E181h) Load 32 bit ptr
93.
94.             db      0FEh
```

sub_04_listing

```
1. sub_4        proc  near
2.
3.              ; -- Сохранение регистров ds, dx
4.              push  ds
5.              push  ax
6.
7.              ; -- AX = DS = 0040H
8.              mov   ax,40h
9.              mov   ds,ax
10.
11.             ; -- Сохранение младшего байта FLAGS в AH
12.             lahf                    ; Load ah from flags
13.
14.             ; -- Проверка флага DF либо старшего бита IOPL
15.             test  word ptr ds:[314h],2400h    ; (0040:0314=3200h)
16.             jnz   loc_9             ; Jump if not zero
17.
18.             ; -- Сброс Interrput Enable Flag, 9 бит занулить
19.               lock      and   word ptr ds:[314h],0FDFFh   ;
    (0040:0314=3200h)
20.   loc_8:
21.             ; -- Загрузка AH в младший байт FLAGS
22.             sahf                    ; Store ah into flags
23.             pop   ax
24.             pop   ds
25.             jmp   short loc_10          ; (07D8)
26.   loc_9:
27.             cli                     ; Disable interrupts
28.             jmp   short loc_8           ; (07D0)
29.   loc_10:
30.             retn
31.   sub_4         endp
```

ЛР2

```
.386p

;1 какую программу вы написали в защищенном режиме (в коде показать пальцем)
программу 0 уровня привелегии в ЗР
0 уровень из-за того, что мы используем таблицы IDT и GDT
;2 какие две системные таблицы нам пришлось создать и почему
Табл. дескр. прер. и табл. дескр. сегментов
GDT и IDT
;3 почему нам пришлось создать две системные таблицы в этой программме
Создание GDT обусловлено необходимостью обращения к сегментам в защищенном режиме, которое возможно исключительно через дескрипторы этих сегментов.
В этой таблице описывается столько дескрипторов, сколько используется в программе.
IDT - для корректного вычисления обработчика прерываний
;4 какие сегменты мы описали в GDT и для чего
32б сегмент кода
32б сегмент данных (4Гб)
32б сегмент данных
32б сегмент стека
16б сегмент видеобуффера
;5 охарактеризовать сегменты, объявленные для подсчета доступного адресного пространства по полям и флагам
gdt_flatDS     descr <0FFFFh,                0,             0,      92h,11001111b,0>
lim     dw 0 ; Граница (биты 0...15) определяет базовый (начальный) 32-разрядный адрес сегмента в физическом адресном пространстве
base_l  dw 0 ; База, биты 0...15
base_m  db 0 ; База, биты 16...23
attr_1  db 0 ; Байт атрибутов 1 // первые 4 бита - тип
attr_2  db 0 ; Граница (биты 16...19) и атрибуты 2
base_h  db 0 ; База, биты 24...31
98h - для сегмента кода, executable, невозможен read-write
 92h - для остальных сегментов, вомзможен read-write
бит дробности - старший бит поля атрибутов. Если установлен, граница интерпретируется в единицах по 4К байт, если сброшен – в байтах.
;6 почему в программе прерывания от таймера и клавиатуры нельзя называть 8h и 9h в этой программе
Потому что мы изменяем контроллер прерырваний. Базовый вектор меняется с 8 на 32 и обратно, получается,
что увеличивается количество с 2^8 до 2^32 кодов.
;7 почему наши таблицы дескрипторов прерываний имеют такую структуру
От 0 до 31 - это исключения (внутренние прерывания).
Дескрипторы в таблице прерываний должны быть расположены по порядку их векторов.
Поэтому мы их все тут по порядку прописываем.
Исключение общей защиты Выделеяем отдельно, чтобы в дальнейшем смогли его отдельно обработать, Т.к. там нужно еще учитывать код ошибки.
        ; 10001110 - 8Eh - шлюз прерываний - служит для обработки прерывания.
int_key int_descr      <0, SEL_32bitCS,0, 8Eh, 0> описаны шлюзы - 8Fh - ловушки 8Eh - прерывание
int_timer int_descr  <0, SEL_32bitCS,0, 8Eh, 0>
;8 что нам пришлось сделать с контроллером прерываний и почему
Пришлось изменить контроллер прерывания
;9 что мы погли использовать в своем обработчике прерываний и почему
Порт клавиатуры
;10 что такое лини A20 и что с ней нужно делать при переходе в защищенный режим
Перед переходом в защищенный режим (или после перехода в него)
следует открыть линию A20, т.е. адресную линию, на которой устанавливается единичный уровень сигнала,
если происходит обращение к мегабайтам адресного пространства с номерами·1, 3, 5 и т.д. (первый
мегабайт имеет номер 0). В реальном режиме линия A20 заблокирована, и если значение адреса выходит за пределы FFFFFh,
выполняется его циклическое оборачивание (линейный адрес 100000h превращается в
00000h, а.црсс 100001h ·.в 00001h и т.д.)
;11 минимальные необходимые действия для возвращения в реальный режим
1. Устанавливаем флаг перех. в рр (PE в управляющем регистре CR0)
2. Запрещаем маскируемые прерывания
3. Переходим в реальный
4. Через команду far jmp, заданную прямо кодом, обновляем значение в теневом регистре, связанном с CS
5. Обновляем остальные теневые регистры значениями
6. Возвращаем маски контроллерам прерываний, возвращаем значение базового вектора прерывания (8, не 32)
7. Возвращаем базовый линейный адрес таблице векторов прерываний
8. Разрешаем немаскируемые
```

```
9. Разрешаем маскируемые
10. Печатаем сообщение с помощью функций Dos
11. Выходим через функцию DOS
; structure for descriptor
descr struc
        lim     dw 0 ; Граница (биты 0...15)
        base_l  dw 0 ; База, биты 0...15
        base_m  db 0 ; База, биты 16...23
        attr_1  db 0 ; Байт атрибутов 1 // первые 4 бита - тип
        attr_2  db 0 ; Граница (биты 16...19) и атрибуты 2
        base_h  db 0 ; База, биты 24...31
descr ends

; structure for interrupt descriptor
int_descr struc
        offs_l  dw 0 ; Смещение обработчика (биты 0...15)
        sel             dw 0 ; Селектор сегмента команд
        counter db 0 ; Зарезервировано
        attr    db 0 ; Атрибуты
        offs_h  dw 0 ; Смещение обработчика (биты 16...31)
int_descr ends

; protected mode segment description
ProtectedSeg   segment para public 'code' use32
            ASSUME cs:ProtectedSeg

        ; creating GDT with params
        ; для описания сегментов физической памяти, с которыми будет работать запущенная программа
        GDT     label byte
        gdt_null        descr <> ; нулевой дескриптор
        gdt_flatDS      descr <0FFFFh,                  0,              0,      92h,11001111b,0>
;92h(read-write) = 10010010b , Селектор 8, сегмент данных
        gdt_16bitCS     descr <RM_seg_size-1, 0,                0,      98h,0           ,0>
;98h(executable, read-write prohibited) = 10011010b
        gdt_32bitCS     descr <ProtectedSeg_size-1,  0,             0,      98h,01000000b,0> ; Селектор 8, сегмент команд , executable
        gdt_32bitDS     descr <ProtectedSeg_size-1,  0,             0,      92h,01000000b,0> ; Селектор 16, сегмент данных
        gdt_32bitSS     descr <stack_len-1,          0,             0,      92h,01000000b,0> ; Селектор 24, сегмент стека
        gdt_32Video     descr <0FFFFh,                  8000h,  0Bh,92H,01001111b,0> ; Селектор 32, видеобуфер, в первом мегабайте адресного пространства
        gdt_size = $ - GDT ; Размер GDT
        gdtr df 0 ; Псевдодескриптор (для lgdt)

        ; creating macros with selectors' offsets
       SEL_flatDS      equ   8
       SEL_16bitCS     equ   16
       SEL_32bitCS     equ   24
       SEL_32bitDS     equ   32
       SEL_32bitSS     equ   40
        SEL_32Video equ   48

        ; Создаю таблицу прерываний IDT
        ; Пропускаю первые 13, 14-е назначаю general exp.
        ; Пропускаю 18, после этого начнется прерывание
        ; после пропущенных 13 + 18 начинается прерывание
       IDT     label byte
       int_descr 13 dup       (<0, SEL_32bitCS,0, 8Fh, 0>) ; 10001111,type - trap
       gen_exp int_descr    <0, SEL_32bitCS,0, 8Fh, 0>
        int_descr 18 dup      (<0, SEL_32bitCS,0, 8Fh, 0>)
       int_timer int_descr  <0, SEL_32bitCS,0, 8Eh, 0>      ; 10001110,type - interrupt
       int_key int_descr      <0, SEL_32bitCS,0, 8Eh, 0>
       idt_size = $ - IDT                                                              ; Размер текущей таблицы $-IDT
       idtr            df 0
       idtr_real       dw 3FFh, 0, 0                                           ; Память под таблицу прерываний реального режима dw 3FFh, 0, 0

       ; Сохраняю маски контроллеров
        ; when exiting protected mode to get original parameters of descriptor table and interrupt table
       master          db 0 ; in stack
       slave           db 0 ; offset for moving in protected mode
```

```
126.
127.        ; Выделение флага escape и сохранение времени
128.        escape          db 0 ; flag that changes when key is pressed
129.        time_save       dd 0 ; flag that shows the program is still on, defaults to zero when
    moving to 4GB + 1 byte
130.
131.        ; Устанавливаю значения выводимых сообщений
132.        msg1 db 'In Real Mode now. To move to Protected Mode press any key...$'
133.        msg2 db 'Back in Real Mode$'
134.        msg3 db 'Protected Mode$'
135.
136.        ; ASCII таблица
137.        ASCII_table     db 0, 0, 49, 50, 51, 52, 53, 54, 55, 56, 57, 48 ; NullEsc1234567890
138.                                db 45, 61, 0, 0
                            ; -=,Backspace,Tab
139.                                db 113, 119, 101, 114, 116, 121
            ; qwerty
140.                                db 117, 105, 111, 112, 91, 93                                   ;
    uiop[]
141.                                db 0, 0
                            ; Enter,Ctrl
142.                                db 97, 115, 100, 102, 103
            ; asdfg
143.                                db 104, 106, 107, 108, 59, 39                                   ;
    hjkl;'-symbol
144.                                db 96, 0, 92
                    ; `LShift,\'-screen
145.                                db 122, 120, 99, 118, 98
            ; zxcvb
146.                                db 110, 109, 44, 46, 47
                    ; nm,./
147.                                db 0, 42, 0, 32
                            ; RShift,*Alt,Space
148.                                db 0
                            ;CapsLock
149.                                db 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0
            ;F1-F10
150.                                db 0, 0, 0, 0, 0                ;
    NumLock,ScrollLock,Home,UpArrow,PgUp
151.                                db 45                                       ; -
152.                                db 0, 0, 0                          ; LeftArrow,CenterKey,RightArrow
153.                                db 43                                       ; +
154.                                db 0, 0, 0, 0, 0                ; End,DownArrow,PgDown,Ins,Del
155.        out_pos         dd 1E0h                                 ; 80 * 2 * 3 -> 3 line in VideoMem
156.        line_curr       dd 1E0h
157.        char_cont    db 0
158.        ; Bright color 4bit - 1
159.        font_col        db 00000010b ;white - 111b, black - 000b,blue - 001b, green - 010b,red -
    100b
160.
161.    num_convert macro
162.        local number1
163.                cmp dl,10
164.                jl number1
165.                add dl,'A' - '0' - 10
166.        number1:
167.                add dl,'0'
168.    endm
169.
170.    print_eax macro
171.        local cycle
172.                push ecx
173.                push dx
174.
175.                mov ecx,8
176.                add esi,10
177.        cycle:
178.                mov dl,al
179.                and dl,0Fh
180.                mov dh,font_col
181.                num_convert
182.
183.                mov es:[esi],dx
184.                ror eax,4
185.                dec esi
```

```
186.              dec esi
187.              loop cycle
188.
189.              pop dx
190.              pop ecx
191.    endm
192.
193.    ; Вход в защищенный режим
194.    PM_entry:
195.        mov     ax,SEL_32bitDS ; Селектор сегмента данных
196.        mov     ds,ax
197.        mov ax,SEL_32Video ; Селектор сегмента видеобуфера
198.        mov es,ax
199.        mov     ax,SEL_flatDS
200.        mov     gs,ax
201.        mov     ax,SEL_32bitSS  ; Селектор сегмента стека
202.        mov     ebx,stack_len
203.        mov     ss,ax
204.        mov     esp,ebx
205.        ; enable interrupts (only our remains)
206.        sti
207.        call memory_counter
208.        ; Вывод сообщения о защищённом режиме работы процессора
209.        mov esi,offset msg3
210.        xor edi,edi
211.        push ecx
212.        push dx
213.        mov dh,font_col ; text color
214.        or dh,00001000b ; bright text
215.        mov ecx,14 ; message len
216.    cycle1: ; printing symbol per iter for 14 times
217.        mov dl,ds:[esi]
218.        mov es:[edi],dx
219.        inc esi
220.        inc edi
221.        inc edi
222.        loop cycle1
223.
224.        pop dx
225.        pop ecx
226.    work: ; idling until key pressed
227.        test escape, 1
228.        jz      work
229.    return:
230.        ; disabling interruptions for safe exiting
231.        cli
232.        ; Far jmp to RM_return (6 byte)
233.        ; Загрузка в CS:IP селектора - смещение точки RM_return, т.к. напрямую в CS грузить
    нельзя
234.        db      0EAh
235.        dd      offset RM_return
236.        dw      SEL_16bitCS
237.
238.    ; Timer interrupt
239.    timer_int:
240.        push eax
241.        push ebp
242.        push ecx
243.        push dx
244.
245.        mov  eax,time_save
246.
247.        xor esi,esi
248.        mov esi,164
249.        print_eax
250.
251.        inc eax
252.        mov time_save,eax
253.
254.        pop dx
255.        pop ecx
256.        pop ebp
257.    ; Отправление команды ведущему контроллеру прерываний
258.        mov     al,20h
259.        out     20h,al
```

```
260.        pop eax
261.        iretd
262.
263.    ; Keyboard interrupt
264.    keyboard_int:
265.        push eax
266.        push ebx
267.        push ebp
268.        push edx
269.    ; Получение скан кода клавиши с порта клавиатуры
270.        in      al,60h
271.
272.        cmp     al,1Ch   ; Enter keycode
273.        jne     stay
274.        mov escape,1
275.        jmp quit
276.    stay:
277.        cmp al,80h ; Проверка что не отжата
278.        ja quit
279.
280.    ; Delete cursor
281.        mov ebx,out_pos
282.        mov dx,0000h
283.        mov dh,font_col
284.        mov es:[ebx],dx
285.
286.        cmp al,0Eh     ; If is backspace keycode
287.        jne print
288.
289.        xor dx,dx
290.        mov dh,font_col
291.        dec ebx
292.        dec ebx
293.
294.        cmp ebx,1E0h
295.        jl      border
296.
297.        mov es:[ebx],dx
298.        mov out_pos,ebx
299.        dec char_cont
300.
301.        cmp char_cont,0
302.        jl backline
303.
304.        jmp quit
305.    border:
306.        mov out_pos,1E0h
307.        jmp quit
308.    backline:
309.        mov ebx,line_curr
310.        sub ebx,160
311.        mov line_curr,ebx
312.        mov char_cont,80
313.        jmp quit
314.    print:
315.        xor ah,ah
316.        mov bp,ax
317.        inc char_cont
318.        cmp char_cont,80
319.        jne continue
320.        mov char_cont,0
321.        add line_curr,160
322.    continue:
323.        mov dl,ASCII_table[ebp]
324.        mov dh, font_col
325.        mov ebx,out_pos
326.        mov es:[ebx],dx
327.        inc ebx
328.        inc ebx
329.        mov out_pos,ebx
330.    quit:
331.        mov ebx,out_pos
332.        mov dl,60
333.        mov dh,10000000b ; 7 - symbol flash, 6-4 - font color, 2-0 - symbol color
334.        or dh,font_col
```

```
335.        mov es:[ebx],dx
336.
337.        in      al,61h  ; Output received
338.        or      al,80h
339.        out     61h,al
340.
341.        mov     al,20h  ; End of Interrupt
342.        out     20h,al
343.        pop edx
344.        pop ebp
345.        pop ebx
346.        pop     eax
347.        iretd
348.
349.
350.    ; General defence exception handler
351.    new_gen_exp:
352.        pop EAX
353.        pop EAX
354.        mov ax, 0Dh
355.        iretd
356.
357.    ; Compute available memory
358.    memory_counter    proc
359.        push    ds
360.        mov     ax, SEL_flatDS ; 4Гб селектор
361.        mov     ds, ax
362.        mov     ebx, 100001h
363.        mov     dl,     10101010b
364.        mov     ecx, 0FFEFFFFEh       ; 1Мб пропускается (BIOS,ROM area) - Оставшиеся  4 гб
365.    check:
366.        mov     dh, ds:[ebx]
367.        mov     ds:[ebx],dl ; Запись сигнатуры
368.        cmp     ds:[ebx],dl ; Сравнение сигнатуры
369.        jnz     mem_end ; Завершает таймер
370.        mov     ds:[ebx],dh
371.        inc     ebx ; Инкремент счетчика
372.        loop check
373.    mem_end:
374.        pop     ds
375.        xor     edx, edx
376.        mov     eax, ebx
377.        mov     ebx, 100000h
378.        div     ebx     ; Для запили в Мб
379.        ; Выход
380.        xor esi,esi
381.        mov esi,182;20
382.        print_eax
383.        ret
384.        memory_counter endp
385.        ; Процессор работает в реальном режиме
386.        ProtectedSeg_size = $ - GDT ; Установка размер таблицы
387.    ProtectedSeg      ENDS
388.
389.    ; used to store data from real mode when quiting it
390.    ; to be able to return back to it with all the same params
391.    SS_seg segment para stack 'stack'
392.        stack_start    db     100h dup(?)
393.        stack_len = $ - stack_start
394.    SS_seg     ENDS
395.
396.    ; Вход в реальный режим
397.    ; Real Mode
398.    RM_seg     segment para public 'code' use16
399.        ASSUME cs:RM_seg, ds:ProtectedSeg, ss:SS_seg
400.    start:
401.
402.        ; Загружаю в сегмент регистры селекторы
403.        mov ax,ProtectedSeg
404.        mov ds,ax ; Загрузка сегментного адреса сегмента данных.
405.    ; Вывод сообщения о реальном режиме работы процессора
406.        mov ah, 09h
407.        mov edx, offset msg1
408.        int 21h
409.
```

```
410.        push eax
411.        mov ah,10h
412.        int 16h ; puts to eax the key pressed in ascii
413.        pop eax
414.
415.        mov     ax,3 ; moving to videomem text mode
416.        int     10h
417.
418.    ; Shadow register setup (uncomment for better perfomance / that calcs register offset)
419.        ;push ProtectedSeg
420.        ;pop ds
421.
422.        ; loading offset of my own segments decalred above to descriptor
423.        ; setting pointer to the end of the memory of GDT and puts into register holding GDT
424.        ; (not to override our GDT table with the new one when returning from real to
    protected)
425.        xor     eax,eax
426.        mov     ax,RM_seg  ; Загружаем сегментный адрес сегмента данных
427.        ; Вычисление 32-битного линейного адреса сегмента данных и загрузка
428.        ; его в дескриптор сегмента данных GDT
429.        shl     eax,4 ; Сдвиг содержимого EAX на 4 байта влево
430.        mov     word ptr gdt_16bitCS.base_l,ax ; Загрузка младшей части базы
431.        shr     eax,16 ; Сдвиг содержимого EAX на 16 байт вправо
432.        mov     byte ptr gdt_16bitCS.base_m,al ; Загрузка средней части базы
433.        mov     ax,ProtectedSeg
434.        shl     eax,4 ; Сдвиг содержимого EAX на 4 байта влево
435.        push eax
436.        push eax
437.        mov     word ptr GDT_32bitCS.base_l,ax ; Загрузка младшей части базы дескриптора
    сегмента кода
438.        mov     word ptr GDT_32bitSS.base_l,ax ; дескриптора сегмета стека
439.        mov     word ptr GDT_32bitDS.base_l,ax ; дескриптора сегмента данных
440.        shr     eax,16 ; Сдвиг содержимого EAX на 16 байт вправо
441.        mov     byte ptr GDT_32bitCS.base_m,al ; Загрузка средней части базы дескриптора
    сегмента кода
442.        mov     byte ptr GDT_32bitSS.base_m,al ; дескриптора сегмета стека
443.        mov     byte ptr GDT_32bitDS.base_m,al ; дескриптора сегмента данных
444.
445.        pop eax
446.        add     eax,offset GDT
447.    ; Подготовка псевдодескриптора
448.        mov     dword ptr gdtr + 2,eax
449.        mov word ptr gdtr, gdt_size - 1
450.        ; Загрузка регистра GDTR
451.        lgdt fword ptr gdtr
452.        pop     eax
453.        add     eax,offset IDT
454.        mov     dword ptr idtr + 2,eax
455.        mov word ptr idtr, idt_size - 1
456.
457.    ; Вычисление адресов новых обработчиков прерываний для замены в контроллере
458.        mov     eax, offset timer_int ;; setting keyboard interrupt - 32 (размерность)
459.        mov     int_timer.offs_l,ax
460.        shr     eax, 16
461.        mov     int_timer.offs_h,ax
462.        mov     eax, offset keyboard_int ; setting keyboard interrupt - 33
463.        mov     int_key.offs_l,ax
464.        shr     eax, 16
465.
466.        mov     int_key.offs_h,ax
467.        mov     eax, offset new_gen_exp ; General defence exception handler
468.        mov     gen_exp.offs_l,ax
469.        shr     eax, 16
470.        mov     gen_exp.offs_h,ax ; general expression (for verification to use only my
    interrupts)
471.    ; Сохранение масок прерываний ведущего и ведомого контроллеров.
472.        in      al, 21h
473.        mov     master,al
474.        in      al, 0A1h
475.        mov     slave,al
476.    ; Изменение базового вектора.
477.    ; allowing interrupts controller to use my own interrupts
478.        mov     al, 11h
479.        out     20h, al
480.        mov     AL, 20h
```

```
481.        out     21h, al
482.        mov     al, 4
483.
484.        out     21h, al
485.        mov     al, 1
486.        out     21h, al
487.
488.    ; Установка новых масок
489.        mov     al, 0FCh
490.        out     21h, al
491.        mov     al, 0FFh
492.        out     0A1h, al
493.    ; Загрузим регистр IDTR
494.        lidt fword ptr idtr
495.
496.    ; Enable A20 line (A20 is the line that enables changing flags for mode changing)
497.    ; Открытие линии A20
498.        in      al,92h
499.        or      al,2
500.        out     92h,al
501.
502.    ; Clear Interrup Flag - no interrupts
503.        cli
504.    ; Turn off NonMaskableInterrupts
505.        in      al,70h
506.        or      al,80h
507.        out     70h,al
508.    ; Enter protected mode
509.    ; Переход в защищённый режим
510.        mov     eax,cr0
511.        or      al,1 ; Установка PE
512.        mov     cr0,eax
513.    ; Far jmp with operand modificator
514.        db      66h
515.        db      0EAh
516.        dd      offset PM_entry
517.        dw      SEL_32bitCS ; where masks are contained
518.
519.    RM_return:
520.    ; changing register's controller bit
521.        mov     eax,cr0
522.        and     al,0FEh ; уствновка бита защищенного режима
523.        mov     cr0,eax
524.    ; Far jmp to "mov ax,ProtectedSeg"
525.        db      0EAh
526.        dw      $ + 4
527.        dw      RM_seg
528.
529.    ; restring segment registers for real mode
530.        mov     ax,ProtectedSeg
531.        mov     ds,ax
532.        mov     es,ax
533.        mov     ax,SS_seg
534.        mov     bx,stack_len
535.        mov     ss,ax
536.        mov     sp,bx
537.
538.    ; Return base vector (8 byte in real and 32 bytes in protectd mode )
539.        mov     al, 11h
540.        out     20h, al
541.        mov     al, 8
542.        out     21h, al
543.        mov     al, 4
544.        out     21h, al
545.        mov     al, 1
546.        out     21h, al
547.
548.    ; Return masks
549.        mov     al, master
550.        out     21h, al
551.        mov     al, slave
552.        out     0A1h, al
553.
554.        lidt fword ptr idtr_real
555.    ; NonMaskableInterrupts on
```

```
556.        in      al,70h
557.        and     al,07FH
558.        out     70h,al ; general interrupts set on
559.    ; Interrupts on
560.        sti
561.        mov     ax,3
562.        int     10h
563.        mov ah, 09h
564.        mov edx, offset msg2
565.        int 21h
566.    ; Disable A20 line
567.        in      al,92h
568.        or      al,2
569.        out     92h,al
570.    ; Exit in RM
571.        mov     ah,4Ch
572.        int     21h
573.    RM_seg_size = $ - start
574.    RM_seg      ENDS
575.    END start
```

ЛР3

```c
#include <stdio.h>
#include <unistd.h>

// unistd
// прик омпиляции для того, чтобы не путаться с названиями программы, название для исполняемого файла не указываем, оно по умолчанию оно будет устанволено a.out
// Для того, чтобы это было выполнено нужно набраться gcc навзания исходника

int main()
{
    // Создаю поток
    int childpid;

    if ((childpid = fork()) == -1)
    {
        perror("Can't fork.\n");
        return 1;
    }
    else if (childpid == 0)
    {
        while (1)
        {
            printf("%d ", getpid());
        }
    }
    //
    else
    {
        while(1)
        {
            printf("%d ", getpid());
        }
    }

    return 0;
}
```

ЛР4 Задание№1

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
        pid_t child_1, child_2;

        if ((child_1 = fork()) == -1)
        {
                perror("Can't fork");
                exit(1);
        }

        else if (child_1 == 0)
        {
                printf("\nПотомок child_1:\
               \nСобственный идентификатор (pid) = %d\
               \nИдентификатор предка (ppid) = %d\
               \nИдентификатор группы = %d\n",
               getpid(), getppid(), getpgrp());
        sleep(2);
        printf("\nПотомок child_1:\
               \nСобственный идентификатор (pid) = %d\
               \nИдентификатор предка (ppid) = %d\
               \nИдентификатор группы = %d\n",
               getpid(), getppid(), getpgrp());
        exit(0);
        }

        if ((child_2 = fork()) == -1)
        {
                perror("Can't fork");
                exit(1);
        }
        else if (child_2 == 0)
        {
        printf("\nПотомок child_2:\
               \nСобственный идентификатор (pid) = %d\
               \nИдентификатор предка (ppid) = %d\
               \nИдентификатор группы = %d\n",
               getpid(), getppid(), getpgrp());
                sleep(2);
        printf("\nПотомок child_2:\
               \nСобственный идентификатор (pid) = %d\
               \nИдентификатор предка (ppid) = %d\
               \nИдентификатор группы = %d\n",
               getpid(), getppid(), getpgrp());
        exit(0);
        }
        printf("\nПредок:\
           \nСобственный идентификатор (pid) = %d\
           \nИдентификатор группы = %d\
           \nИдентификатор потомка child_1 (pid) = %d\
           \nИдентификатор потомка child_2 (pid) = %d\n",
           getpid(), getpgrp(), child_1, child_2);
        return 0;
}
```

ЛР4 Задание№2

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main()
{
        pid_t child_1, child_2;
        int status, val;

        if ((child_1 = fork()) == -1)
        {
                perror("Fork failed");
                exit(1);
        }

        else if (child_1 == 0)
        {
        printf("\nПотомок child_1:\
                \nСобственный идентификатор (pid) = %d\
                \nИдентификатор предка (ppid) = %d\
                \nИдентификатор группы = %d\n",
                getpid(), getppid(), getpgrp());
                exit(0);
        }

        if ((child_2 = fork()) == -1)
        {
                perror("Fork failed");
                exit(1);
        }
        else if (child_2 == 0)
        {
        printf("\nПотомок child_2:\
                \nСобственный идентификатор (pid) = %d\
                \nИдентификатор предка (ppid) = %d\
                \nИдентификатор группы = %d\n",
                getpid(), getppid(), getpgrp());
                exit(0);
        }

   while ((val = wait(&status)) != -1) {
        if (WIFEXITED(status))
            printf("\nДочерний процесс (%d) завершён нормально с кодом (%d).\n",
                   val, WEXITSTATUS(status));
        else if (WIFSIGNALED(status))
            printf("\nДочерний процесс (%d) завершён неперехватываемым сигналом №(%d)\n",
val, WTERMSIG(status));
        else if (WIFSTOPPED(status))
            printf("\nДочерний процесс (%d) остановился, номер сигнала: (%d)\n", val,
WSTOPSIG(status));
    }

   printf("\nПредок:\
          \nСобственный идентификатор (pid) = %d\
          \nИдентификатор группы = %d\
          \nИдентификатор потомка child_1 (pid) = %d\
          \nИдентификатор потомка child_2 (pid) = %d\n",
          getpid(), getpgrp(), child_1, child_2);
        return 0;
}
```

ЛР4 Задание№3

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define NUMBER_OF_CHILDREN 2

int main()
{
	char *p_exec[NUMBER_OF_CHILDREN] = {
		"meta/average/a.out",
		"meta/factorial/a.out"
	};

	printf("\nРодитель parent:\
		\nСобственный идентификатор (pid) = %d\
		\nИдентификатор группы = %d\n",
		getpid(), getpgrp());

	for (size_t i = 0; i < NUMBER_OF_CHILDREN; i++) {
		pid_t child;

		if ((child = fork()) == -1)
		{
			perror("Can't fork");
			exit(1);
		}

		else if (child == 0)
		{
		 printf("\nПотомок child_%ld: \
			\nСобственный идентификатор (pid) = %d\
			\nИдентификатор предка (ppid) = %d\
			\nИдентификатор группы = %d\n",
			i, getpid(), getppid(), getpgrp());

			if (execl(p_exec[i], NULL) == -1)
			{
				perror("exec");
				exit(2);
			}

		}
	}

	for (size_t i = 0; i < NUMBER_OF_CHILDREN; i++) {
		int status = 0;
		int val = 0;

		pid_t childpid = wait(&status);

		printf("\nРодитель parent: child %ld\
		\nСобственный идентификатор (pid) = %d\
		\nСтатус = %d\n",
		i + 1, childpid, status);

		if (WIFSIGNALED(val))
		{
			printf("\nРодитель parent: child %ld\
			 \nКод окончания: %d\n", i + 1, WTERMSIG(val));
		}
		else if (WIFEXITED(val))
		{
			printf("\nРодитель parent: child %ld\
			 \nКод окончания: %d\n", i + 1, WEXITSTATUS(val));
		}
		else if (WIFSTOPPED(val))
		{
			printf("\nРодитель parent: child %ld\
			 \nКод окончания: %d\n", i + 1, WSTOPSIG(val));
		}
	}

	return 0;
```

```
74. }
```

ЛР4 Задание№4

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

int main()
{
        pid_t child_1, child_2;
    int status, val, fd[2];

        if (pipe(fd) == -1)
        {
                perror("Pipe failed");
                exit(1);
        }
        if ((child_1 = fork()) == -1)
        {
                perror("Fork failed");
                exit(2);
        }
        else if (child_1 == 0)
        {
        printf("\nПотомок child_1:\
               \nСобственный идентификатор (pid) = %d\
               \nИдентификатор предка (ppid) = %d\
               \nИдентификатор группы = %d\n",
               getpid(), getppid(), getpgrp());
        char message[] = "kdjfskjfhdksfhdsjhbncsdlkvjbf";
                close(fd[0]);
                write(fd[1], message, sizeof message - 1);
                printf("\nПотомок child_1 написал: %s\n", message);
                exit(0);
        }

        if ((child_2 = fork()) == -1)
        {
                perror("Fork failed");
                exit(3);
        }
        else if (child_2 == 0)
        {
        printf("\nПотомок child_2:\
               \nСобственный идентификатор (pid) = %d\
               \nИдентификатор предка (ppid) = %d\
               \nИдентификатор группы = %d\n",
               getpid(), getppid(), getpgrp());
        char message[] = "209482409384092834902384209348209348423490283420938423094823049 8";
                close(fd[0]);
                write(fd[1], message, sizeof message - 1);
        printf("\nПотомок child_2 написал: %s\n", message);
                exit(0);
        }

    while ((val = wait(&status)) != -1) {
        if (WIFEXITED(status))
            printf("\nДочерний процесс (%d) завершён нормально с кодом (%d).\n",
                   val, WEXITSTATUS(status));
        else if (WIFSIGNALED(status))
            printf("\nДочерний процесс (%d) завершён неперехватываемым сигналом №(%d)\n",
    val, WTERMSIG(status));
        else if (WIFSTOPPED(status))
            printf("\nДочерний процесс (%d) остановился, номер сигнала: (%d)\n", val,
    WSTOPSIG(status));
    }

    printf("\nПредок:\
           \nСобственный идентификатор (pid) = %d\
           \nИдентификатор группы = %d\
           \nИдентификатор потомка child_1 (pid) = %d\
           \nИдентификатор потомка child_2 (pid) = %d\n",
           getpid(), getpgrp(), child_1, child_2);

    close(fd[1]);
```

```
73.     char message[100];
74.     read(fd[0], message, sizeof message);
75.     printf("\nПредок прочитал: %s\n", message);
76.
77.         return 0;
78. }
```

ЛР4 Задание№5

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

int mode = 0;

void quit_signal_handler(int signum)
{
   mode = 1;
}

int main()
{
        int val, status, fd[2];
   pid_t child_1, child_2;

        if (pipe(fd) == -1)
        {
        perror("Pipe failed");
                exit(1);
        }

   signal(SIGQUIT, quit_signal_handler);

   if ((child_1 = fork()) == -1)
   {
       perror("Fork failed");
       exit(2);
   }
   else if (child_1 == 0)
   {
       printf("\nПотомок child_1:\
              \nСобственный идентификатор (pid) = %d\
              \nИдентификатор предка (ppid) = %d\
              \nИдентификатор группы = %d\n",
              getpid(), getppid(), getpgrp());
       signal(SIGQUIT, quit_signal_handler);
       sleep(6);
       if (mode == 1)
       {
           char message[] = "(Сообщение от child_1)";
           close(fd[0]);
           write(fd[1], message, sizeof message - 1);
           printf("\nСигнал (Ctrl-\\) пришёл.");
           printf("\nПотомок child_1 написал: %s\n", message);
       }
       else
       {
           printf("\nСигнал (Ctrl-\\) не пришёл.\n");
       }
       exit(0);
   }

   if ((child_2 = fork()) == -1)
   {
       perror("Fork failed");
       exit(3);
   }
   else if (child_2 == 0)
   {
       printf("\nПотомок child_2:\
              \nСобственный идентификатор (pid) = %d\
              \nИдентификатор предка (ppid) = %d\
              \nИдентификатор группы = %d\n",
              getpid(), getppid(), getpgrp());
       signal(SIGQUIT, quit_signal_handler);
       sleep(6);
       if (mode == 1)
       {
           char message[] = "(Сообщение от child_2)";
           close(fd[0]);
           write(fd[1], message, sizeof message - 1);
           printf("\nСигнал (Ctrl-\\) пришёл.");
```

```
75.             printf("\nПотомок child_2 написал: %s\n", message);
76.         }
77.         else
78.         {
79.             printf("\nСигнал (Ctrl-\\) не пришёл.\n");
80.         }
81.         exit(0);
82.     }
83.
84.     while ((val = wait(&status)) != -1) {
85.         if (WIFEXITED(status))
86.             printf("\nДочерний процесс (%d) завершён нормально с кодом (%d).\n",
87.                    val, WEXITSTATUS(status));
88.         else if (WIFSIGNALED(status))
89.             printf("\nДочерний процесс (%d) завершён неперехватываемым сигналом №(%d)\n",
    val, WTERMSIG(status));
90.         else if (WIFSTOPPED(status))
91.             printf("\nДочерний процесс (%d) остановился, номер сигнала: (%d)\n", val,
    WSTOPSIG(status));
92.     }
93.
94.     printf("\nПредок:\
95.             \nСобственный идентификатор (pid) = %d\
96.             \nИдентификатор группы = %d\
97.             \nИдентификатор потомка child_1 (pid) = %d\
98.             \nИдентификатор потомка child_2 (pid) = %d\n",
99.             getpid(), getpgrp(), child_1, child_2);
100.
101.        close(fd[1]);
102.        char message[100];
103.        read(fd[0], message, sizeof message);
104.        printf("\nПредок прочитал: %s\n", message);
105.
106.         return 0;
107.     }
```

ЛР5 Часть 1

```
1.  #include <sys/shm.h>
2.  #include <sys/sem.h>
3.  #include <fcntl.h>
4.  #include <unistd.h>
5.  #include <sys/wait.h>
6.  #include <stdio.h>
7.  #include <time.h>
8.  #include <stdlib.h>
9.  #include <signal.h>
10.
11. #define SE 0
12. #define SF 1
13. #define SB 2
14.
15. const int buffer_size = 10;
16. const int permissions = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
17. const int consumer_num = 3;
18. const int producer_num = 3;
19.
20. struct sembuf semops_produce_begin[2] = {
21.         {SE, -1, 0},
22.         {SB, -1, 0}
23. };
24.
25. struct sembuf semops_produce_end[2] = {
26.         {SB, 1, 0},
27.         {SF, 1, 0}
28. };
29.
30. struct sembuf semops_consume_begin[2] = {
31.         {SF, -1, 0},
32.         {SB, -1, 0}
33. };
34.
35. struct sembuf semops_consume_end[2] = {
36.         {SB, 1, 0},
37.         {SE, 1, 0}
38. };
39.
40. int** shm_cons_pos;
41. int** shm_prod_pos;
42. int* shm_buffer;
43.
44. pid_t* pid_arr;
45.
46.
47. void kill_producers();
48.
49. void kill_consumers();
50.
51. int* get_shm();
52.
53. int get_sem_fd();
54.
55. int produce(int sem_fd, int number);
56. int consume(int sem_fd, int number);
57.
58. void get_producer(int number, int sem_fd);
59. void get_consumer(int number, const int sem_fd);
60.
61.
62. int main()
63. {
64.         int *shm = get_shm();
65.
66.     shm_cons_pos = shm;
67.         shm_prod_pos = shm + 2;
68.     *(shm + 4) = 0;
69.         shm_buffer = shm + 5;
70.     pid_arr = shm + 5 + buffer_size;
71.
72.         *shm_cons_pos = shm_buffer;
73.         *shm_prod_pos = shm_buffer;
74.
```

```
75.         int sem_fd = get_sem_fd();
76.
77.         int semctl_status = semctl(sem_fd, SE, SETVAL, consumer_num + producer_num);
78.
79.         if (semctl_status != -1)
80.                 semctl_status = semctl(sem_fd, SF, SETVAL, 0);
81.         if (semctl_status != 1)
82.                 semctl_status = semctl(sem_fd, SB, SETVAL, 1);
83.
84.         if (semctl_status == -1)
85.         {
86.                 perror("Semctl failed");
87.                 exit(1);
88.         }
89.
90.     int i = 0;
91.     while(i < producer_num)
92.     {
93.         get_producer(i, sem_fd);
94.         i++;
95.     }
96.
97.         int j = producer_num;
98.         while(j < producer_num + consumer_num)
99.         {
100.                get_consumer(j, sem_fd);
101.                j++;
102.        }
103.
104.        for (int i = 0; i < consumer_num + producer_num; i++)
105.        {
106.            int *status;
107.            if (wait(status) == -1)
108.            {
109.                printf("Waitpid");
110.                exit(9);
111.            }
112.            if (WIFEXITED(status))
113.            {
114.                printf("\tChild exited with code = %d\n", WEXITSTATUS(status));
115.            }
116.            else if (WIFSIGNALED(status))
117.            {
118.                printf("\tChild with (signal %d) killed\n", WTERMSIG(status));
119.            }
120.            else if (WIFSTOPPED(status))
121.            {
122.                printf("\tChild with (signal %d) stopped\n", WSTOPSIG(status));
123.            }
124.            else
125.                printf("\tChild terminated abnormally \n");
126.        }
127.
128.        if (shmdt(shm) == -1)
129.        {
130.                perror("Shmdt failed.\n");
131.            exit(10);
132.        }
133.
134.        return 0;
135.    }
136.
137.
138.    void kill_producers()
139.    {
140.        for (int i = 0; i < producer_num; i++) {
141.            if (pid_arr[i] == getpid()) {
142.                continue;
143.            }
144.            kill(pid_arr[i], SIGTERM);
145.        }
146.        exit(0);
147.    }
148.
149.    void kill_consumers()
```

```
150.    {
151.        for (int i = producer_num; i < producer_num + consumer_num; i++) {
152.            if (pid_arr[i] == getpid()) {
153.                continue;
154.            }
155.            kill(pid_arr[i], SIGTERM);
156.        }
157.        kill(getpid(), SIGKILL);
158.    }
159.
160.    int* get_shm()
161.    {
162.        int shm_id;
163.        int* shm;
164.
165.        shm_id = shmget(100, sizeof(int) * (buffer_size + 1) + (producer_num + consumer_num)
    * sizeof(pid_t) + sizeof(int*) * 2, IPC_CREAT|permissions);
166.
167.        if (shm_id == -1)
168.        {
169.                perror("Shmget failed.\n");
170.                exit(1);
171.        }
172.
173.        shm = (int*) shmat(shm_id, NULL, 0);
174.        if (*shm == -1)
175.        {
176.                perror("Shmat failed.\n");
177.                exit(2);
178.        }
179.
180.        return shm;
181.    }
182.
183.    int get_sem_fd()
184.    {
185.        int sem_fd = semget(101, 3, IPC_CREAT | permissions);
186.        if (sem_fd == -1)
187.        {
188.            perror("Semget failed.\n");
189.            exit(3);
190.        }
191.        return sem_fd;
192.    }
193.
194.    int produce(int sem_fd, int number)
195.    {
196.        srand(time(NULL));
197.        sleep(rand() % 5);
198.
199.        if (semop(sem_fd, semops_produce_begin, 2) == -1)
200.        {
201.            perror("Semop failed.\n");
202.            exit(4);
203.        }
204.        int **shm_prod_pos = shm_buffer - 3;
205.        int *prod_pos = *shm_prod_pos;
206.
207.        if (*(prod_pos - 1) >= buffer_size)
208.        {
209.            if (semop(sem_fd, semops_produce_end, 2) == -1)
210.            {
211.                perror("Semop failed.\n");
212.                exit(5);
213.            }
214.            kill_producers();
215.            return 0;
216.        }
217.
218.        *prod_pos = *(prod_pos - 1) + 1;
219.        printf("Производитель %d, pid=%d создал значение %d\n", number, getpid(), *prod_pos);
220.        prod_pos++;
221.        *shm_prod_pos = prod_pos;
222.
223.        if (semop(sem_fd, semops_produce_end, 2) == -1)
```

```
224.        {
225.            perror("Semop failed.\n");
226.            exit(6);
227.        }
228.
229.        return 0;
230.    }
231.
232.    int consume(int sem_fd, int number)
233.    {
234.        srand(time(NULL));
235.        sleep(rand() % 5 + 5);
236.
237.        if (semop(sem_fd, semops_consume_begin, 2) == -1)
238.        {
239.            perror("Semop failed.\n");
240.            exit(7);
241.        }
242.        int **shm_cons_pos = shm_buffer - 5;
243.        int *cons_pos = *shm_cons_pos;
244.        if (cons_pos - shm_buffer >= buffer_size)
245.        {
246.            if (semop(sem_fd, semops_consume_end, 2) == -1)
247.            {
248.                perror("Semop failed.\n");
249.                exit(1);
250.            }
251.            kill_consumers();
252.            return 0;
253.        }
254.        printf("Потребитель %d, pid=%d прочитал значение: %d\n", number, getpid(),
    *cons_pos);
255.        cons_pos++;
256.        *shm_cons_pos = cons_pos;
257.
258.        if (semop(sem_fd, semops_consume_end, 2) == -1)
259.        {
260.            perror("Semop failed.\n");
261.            exit(8);
262.        }
263.
264.        return 0;
265.    }
266.
267.    void get_producer(int number, int sem_fd) {
268.        pid_t pid;
269.        if ((pid = fork()) == -1)
270.        {
271.            printf("Fork failed.\n");
272.            exit(4);
273.        }
274.
275.        if(pid == 0)
276.        {
277.            printf("Создан производитель %d, pid: %d\n", number, getpid());
278.            while(1)
279.                produce(sem_fd, number);
280.        }
281.        else
282.        {
283.            pid_arr[number] = pid;
284.        }
285.    }
286.
287.    void get_consumer(int number, const int sem_fd) {
288.        pid_t pid;
289.
290.        if ((pid = fork()) == -1)
291.        {
292.            printf("Fork failed.\n");
293.            exit(1);
294.        }
295.
296.        if(pid == 0)
297.        {
```

```
298.            printf("Создан потребитель %d, pid: %d\n", number, getpid());
299.            while(1)
300.                consume(sem_fd, number);
301.        }
302.        else
303.        {
304.            pid_arr[number] = pid;
305.        }
306.    }
```

ЛР5 Часть 2

```c
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

#define ACTIVE_WRITER        0
#define WAITING_WRITERS      1
#define ACTIVE_READERS       2
#define WAITING_READERS 3

#define WRITERS_NUMBER       3
#define READERS_NUMBER       5


struct sembuf start_read[5] = {{WAITING_READERS, 1, SEM_UNDO},
                               {WAITING_WRITERS, 0, SEM_UNDO},
                               {ACTIVE_WRITER, 0, SEM_UNDO},
                               {WAITING_READERS, -1, SEM_UNDO},
                               {ACTIVE_READERS, 1, SEM_UNDO}};

struct sembuf stop_read[1] = {{ACTIVE_READERS, -1, SEM_UNDO}};

struct sembuf start_write[5] = {{WAITING_WRITERS, 1, SEM_UNDO},
                                {ACTIVE_WRITER, 0, SEM_UNDO},
                                {ACTIVE_READERS, 0, SEM_UNDO},
                                {WAITING_WRITERS, -1, SEM_UNDO},
                                {ACTIVE_WRITER, 1, SEM_UNDO}};

struct sembuf stop_write[1] = {{ACTIVE_WRITER, -1, SEM_UNDO}};

int *shm;

void writer(const int semid, const int index);
void reader(const int semid, const int index);

int main()
{
        int semid, shmid;
        int ctl_ar, ctl_wr, ctl_aw, ctl_ww;
        int status;
        pid_t pid[WRITERS_NUMBER + READERS_NUMBER];
        const int PERMS = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;

        if ((semid = semget(IPC_PRIVATE, 4, IPC_CREAT | PERMS)) == -1)
    {
        perror("Semget failed.\n");
            exit(1);
    }

        ctl_ar = semctl(semid, ACTIVE_READERS, SETVAL, 0);
    ctl_wr = semctl(semid, WAITING_READERS, SETVAL, 0);
    ctl_aw = semctl(semid, ACTIVE_WRITER, SETVAL, 0);
    ctl_ww = semctl(semid, WAITING_WRITERS, SETVAL, 0);

        int ctl_status = ctl_ar;

        if (ctl_status != -1)
                ctl_status = ctl_wr;
        if (ctl_status != -1)
                ctl_status = ctl_aw;
        if (ctl_status != -1)
                ctl_status = ctl_ww;

        if (ctl_status == -1)
        {
                perror("Semget failed.\n");
                exit(2);
        }

        if ((shmid = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | PERMS)) == -1)
```

```
75.    {
76.        perror("Shmget failed.\n");
77.        exit(5);
78.    }
79.
80.        shm = shmat(shmid, 0, 0);
81.        if (*shm == - 1)
82.        {
83.                perror("Shmat failed.\n");
84.                exit(3);
85.        }
86.
87.        (*shm) = 0;
88.
89.        for (int i = 0; i < WRITERS_NUMBER; i++)
90.        {
91.                pid[i] = fork();
92.                if (pid[i] == -1)
93.                {
94.                        perror("Fork failed.\n");
95.                        exit(4);
96.                }
97.                if (pid[i] == 0)
98.                {
99.                        while(1)
100.                               writer(semid, i);
101.                       return 0;
102.               }
103.       }
104.
105.       for (int i = WRITERS_NUMBER; i < WRITERS_NUMBER + READERS_NUMBER; i++)
106.       {
107.               pid[i] = fork();
108.               if (pid[i] == -1)
109.               {
110.                       perror("Fork failed.\n");
111.                       exit(7);
112.               }
113.               if (pid[i] == 0)
114.               {
115.                       while(1)
116.                               reader(semid, i);
117.                       return 0;
118.               }
119.       }
120.
121.       while (wait(&status) != -1){}
122.
123.       if (shmdt(shm) == -1)
124.       {
125.               perror("Shmdt failed.\n");
126.               exit(10);
127.       }
128.
129.       return 0;
130.   }
131.
132.   void writer(const int semid, const int index)
133.   {
134.       srand(time(NULL));
135.       sleep(rand() % 10);
136.       if (semop(semid, start_write, 5) == -1)
137.       {
138.               perror("Semop failed.\n");
139.               exit(5);
140.       }
141.
142.       (*shm)++;
143.
144.       printf("Писатель %d написал %d\n", index + 1, *shm);
145.
146.       if (semop(semid, stop_write, 1) == -1)
147.       {
148.               perror("Semop failed.\n");
149.               exit(6);
```

```
150.        }
151.    }
152.
153.    void reader(const int semid, const int index)
154.    {
155.        srand(time(NULL));
156.        sleep(rand() % 10);
157.        if (semop(semid, start_read, 5) == -1)
158.        {
159.                perror("Semop failed.\n");
160.                exit(8);
161.        }
162.
163.        (*shm)++;
164.
165.        printf("Читатель %d прочитал %d\n", index - WRITERS_NUMBER + 1, *shm);
166.
167.        if (semop(semid, stop_read, 1) == -1)
168.        {
169.                perror("Semop failed.\n");
170.                exit(9);
171.        }
172.    }
```

ЛР6

```
1.  // Надо использовать неделимые операции:
2.  // InterLockedIncrement, InterLockedDecrement.
3.  // В программе должно быть 3 счетчика:
4.  // ждущих писателей, ждущих читателей и активных читателей.
5.  // Активный писатель м.б. только один и это логический тип.
6.
7.  #include <windows.h>
8.  #include <stdbool.h>
9.  #include <stdio.h>
10. #include <time.h>
11. #include <stdbool.h>
12.
13. #define OK 0
14.
15. #define CREATE_MUTEX_ERROR 1
16. #define CREATE_EVENT_ERROR 2
17. #define CREATE_READER_THREAD_ERROR 3
18. #define CREATE_WRITER_THREAD_ERROR 3
19.
20. #define MINIMUM_READER_DELAY 100
21. #define MINIMUM_WRITER_DELAY 100
22. #define MAXIMUM_READER_DELAY 200
23. #define MAXIMUM_WRITER_DELAY 200
24.
25. #define READERS_NUMBER 3
26. #define WRITERS_NUMBER 3
27.
28. #define ITERATIONS_NUMBER 5
29.
30. HANDLE canRead;
31. HANDLE canWrite;
32. HANDLE mutex;
33.
34. LONG waitingWritersCount = 0;
35. LONG waitingReadersCount = 0;
36. LONG activeReadersCount = 0;
37. bool writing = false;
38.
39. HANDLE readerThreads[READERS_NUMBER];
40. HANDLE writerThreads[WRITERS_NUMBER];
41.
42. int readersID[READERS_NUMBER];
43. int writersID[WRITERS_NUMBER];
44.
45. int readersRand[READERS_NUMBER * ITERATIONS_NUMBER];
46. int writersRand[READERS_NUMBER * ITERATIONS_NUMBER];
47.
48. int value = 0;
49.
50. bool turn(HANDLE event)
51. {
52.         // Если функция возвращает WAIT_OBJECT_0, объект свободен.
53.         return WaitForSingleObject(event, 0) == WAIT_OBJECT_0;
54. }
55.
56. void StartRead()
57. {
58.         // Увеличиваем кол-во ждущих читателей.
59.         InterlockedIncrement(&waitingReadersCount);
60.
61.         // Процесс читатель сможет начать работать,
62.         // Если есть нет активного писателя,
63.         // И нет писателей, ждущих свою очередь.
64.         if (writing || turn(canWrite))
65.                 WaitForSingleObject(canRead, INFINITE);
66.
67.         WaitForSingleObject(mutex, INFINITE);
68.         // Уменьшаем кол-во ждущих читателей.
69.         InterlockedDecrement(&waitingReadersCount);
70.         // Увеличиваем кол-во активных читателей.
71.         InterlockedIncrement(&activeReadersCount);
72.         // Выдаем сигнал canRead,
73.         // Чтобы следующий читатель в очереди
74.         // Читателей смог начать чтение
```

```
75.         SetEvent(canRead);
76.         ReleaseMutex(mutex);
77. }
78.
79. void StopRead()
80. {
81.         // Уменьшаем количество активных читателей.
82.         InterlockedDecrement(&activeReadersCount);
83.         // Если число читателей равно нулю,
84.         // Выполняется signal(can_write),
85.         // активизирующий писателя из очереди писателей.
86.         if (!activeReadersCount)
87.                 SetEvent(canWrite);
88. }
89.
90. DWORD WINAPI Reader(CONST LPVOID param)
91. {
92.         int id = *(int *)param;
93.         int sleepTime;
94.         int begin = id * ITERATIONS_NUMBER;
95.         for (int i = 0; i < ITERATIONS_NUMBER; i++)
96.         {
97.                 sleepTime = readersRand[begin + i];
98.                 StartRead();
99.                 printf("Reader with id = %d; value = %d; sleep time = %d.\n", id, value,
    sleepTime);
100.                StopRead();
101.
102.                // WaitForSingleObject(canRead, INFINITE);
103.                // printf("Thread with id = %d, i = %d value = %d\n", id, i, value);
104.                Sleep(sleepTime);
105.        }
106.    }
107.
108.    void StartWrite()
109.    {
110.        // Увеличиваем кол-во ждущих писателей.
111.        InterlockedIncrement(&waitingWritersCount);
112.
113.        // Процесс писатель сможет начать работать,
114.        // Если нет читающих процессов
115.        // И нет другого активного писателя.
116.        if (activeReadersCount > 0 || writing)
117.                WaitForSingleObject(canWrite, INFINITE);
118.
119.        // Уменьшаем кол-во ждущих писателей.
120.        InterlockedDecrement(&waitingWritersCount);
121.        // Писатель пишет.
122.        writing = true;
123.    }
124.
125.    void StopWrite()
126.    {
127.        writing = false;
128.        // Предпочтение отдается читателям при условии,
129.        // Что очередь ждущих читателей не пуста.
130.        if (waitingReadersCount)
131.                SetEvent(canRead);
132.        else
133.                SetEvent(canWrite);
134.    }
135.
136.    DWORD WINAPI Writer(CONST LPVOID param)
137.    {
138.        int id = *(int *)param;
139.        int sleepTime;
140.        int begin = id * ITERATIONS_NUMBER;
141.        for (int i = 0; i < ITERATIONS_NUMBER; i++)
142.        {
143.                sleepTime = writersRand[begin + i];
144.
145.                StartWrite();
146.                ++value;
147.                printf("Writer with id = %d; value = %d; sleep time = %d.\n", id, value,
    sleepTime);
```

```
148.                StopWrite();
149.
150.                Sleep(sleepTime);
151.        }
152.    }
153.
154.    int InitHandles()
155.    {
156.        // 2ой аргумент == false значит мьютекс свободный.
157.        if ((mutex = CreateMutex(NULL, FALSE, NULL)) == NULL)
158.        {
159.                perror("CreateMutex");
160.                return CREATE_MUTEX_ERROR;
161.        }
162.
163.        // 2ой аргумент == FALSE значит автоматический сброс.
164.        // 3ий аргумент == FALSE значит, что объект не в сигнальном состоянии.
165.        if ((canRead = CreateEvent(NULL, FALSE, FALSE, NULL)) == NULL)
166.        {
167.                perror("CreateEvent (canRead)");
168.                return CREATE_EVENT_ERROR;
169.        }
170.
171.        if ((canWrite = CreateEvent(NULL, FALSE, FALSE, NULL)) == NULL)
172.        {
173.                perror("CreateEvent (canWrite)");
174.                return CREATE_EVENT_ERROR;
175.        }
176.
177.        return OK;
178.    }
179.
180.    int CreateThreads()
181.    {
182.        DWORD id = 0;
183.        for (int i = 0; i < READERS_NUMBER; i++)
184.        {
185.                readersID[i] = i;
186.                // Параметры слева направо:
187.                // NULL - Атрибуты защиты определены по умолчанию;
188.                // 0 - размер стека устанавливается по умолчанию;
189.                // Reader - определяет адрес функции потока, с которой следует начать
    выполнение потока;
190.                // readersID + i - указатель на переменную, которая передается в поток;
191.                //  0 - исполнение потока начинается немедленно;
192.                // Последний - адрес переменной типа DWORD, в которую функция возвращает
    идентификатор потока.
193.                if ((readerThreads[i] = CreateThread(NULL, 0, &Reader, readersID + i, 0,
    &id)) == NULL)
194.                {
195.                        perror("CreateThread (reader)");
196.                        return CREATE_READER_THREAD_ERROR;
197.                }
198.                // printf("Created reader with thread id = %d\n", id);
199.        }
200.
201.        for (int i = 0; i < WRITERS_NUMBER; i++)
202.        {
203.                writersID[i] = i;
204.                if ((writerThreads[i] = CreateThread(NULL, 0, &Writer, writersID + i, 0,
    &id)) == NULL)
205.                {
206.                        perror("CreateThread (writer)");
207.                        return CREATE_WRITER_THREAD_ERROR;
208.                }
209.                // printf("Created writer with thread id = %d\n", id);
210.        }
211.
212.        return OK;
213.    }
214.
215.    void Close()
216.    {
217.        // Закрываем дескрипторы mutex, event и всех созданных потоков.
218.        for (int i = 0; i < READERS_NUMBER; i++)
```

```
219.              CloseHandle(readerThreads[i]);
220.
221.       for (int i = 0; i < WRITERS_NUMBER; i++)
222.              CloseHandle(writerThreads[i]);
223.
224.       CloseHandle(canRead);
225.       CloseHandle(canWrite);
226.       CloseHandle(mutex);
227.   }
228.
229.   void CreateRand()
230.   {
231.       for (int i = 0; i < READERS_NUMBER * ITERATIONS_NUMBER; i++)
232.              readersRand[i] = rand() % (MAXIMUM_READER_DELAY - MINIMUM_READER_DELAY) +
    MINIMUM_READER_DELAY;
233.
234.       for (int i = 0; i < WRITERS_NUMBER * ITERATIONS_NUMBER; i++)
235.              writersRand[i] = rand() % (MAXIMUM_WRITER_DELAY - MINIMUM_WRITER_DELAY) +
    MINIMUM_WRITER_DELAY;
236.   }
237.
238.   int main(void)
239.   {
240.       setbuf(stdout, NULL);
241.       srand(time(NULL));
242.
243.       CreateRand();
244.
245.       int err = InitHandles();
246.       if (err)
247.              return err;
248.
249.       err = CreateThreads();
250.       if (err)
251.              return err;
252.
253.       // READERS_NUMBER - кол-во инетерсующих нас объектов ядра.
254.       // readerThreads - указатель на массив описателей объектов ядра.
255.       // TRUE - функция не даст потоку возобновить свою работу, пока не освободятся все
    объекты.
256.       // INFINITE - указывает, сколько времени поток готов ждать освобождения объекта.
257.       WaitForMultipleObjects(READERS_NUMBER, readerThreads, TRUE, INFINITE);
258.       WaitForMultipleObjects(WRITERS_NUMBER, writerThreads, TRUE, INFINITE);
259.
260.       Close();
261.
262.       printf("\nOk!\n");
263.       return OK;
264.   }
```