Министерство образования Республики Беларусь  
Учреждение образования   
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по практической работе №2

По дисциплине «Архитектура вычислительных систем»

По теме

«МУЛЬТИЗАДАЧНОСТЬ В ЗАЩИЩЕННОМ РЕЖИМЕ»

Выполнил:  
студент гр. 253505  
 Сенько Н.С.

Проверил:

Преподаватель

Калиновская Анастасия Александровна

Минск 2024

# Цель работы

Изучить принципы и средства реализации мультизадачности в защищенном режиме процессора. Получить практические навыки по программированию и использованию этих средств.

**Постановка задачи**

Написать программу, реализующую мультизадачность в защищенном режиме. Программа должна переключить процессор в защищенный режим, а затем запустить на выполнение 2-3 задачи, которые должны выполняться параллельно. Каждая задача выводит на экран свое сообщение. Задача выводит на экран часть сообщения, затем происходит переключение на другую задачу и т.д. Когда все задачи отработают программа должна вернуть процессор в реальный режим.

**Практическая работа**

Листинг кода практической работы см. Приложение 1.

Перед тем как переключить процессор в защищенный режим, надо выполнить некоторые подготовительные действия, а именно:

1. Подготовить в оперативной памяти глобальную дескрипторную таблицу.
2. Запретить все маскируемые и немаскируемые прерывания.
3. Запомнить в оперативной памяти содержимое всех сегментных регистров, которые необходимо сохранить для возврата в реальный режим, в частности указатель стека реального режима.
4. Загрузить регистр GDTR.

Для переключение из защищенного режима в реальный режим программа должна выполнить следующие действия:

1. Передать управление сегменту кода с пределом 64 KБ.
2. Загрузить в регистры SS, DS, ES, FS и GS селекторы дескрипторов, подготовленных для адресации памяти в реальном режиме и содержащих соответствующие реальному режиму значения.
3. Запретить все маскируемые и не маскируемые прерывания. Сбросить бит PE регистра CR0, переключив процессор в реальный режим.
4. Выполнить команду дальнего перехода для очистки внутренней очереди команд процессора.
5. Разрешить прерывания.
6. Загрузить в сегментные регистры значения, необходимые для работы в реальном режиме.

При переключении задач процессор выполняет следующие действия:

* + 1. Выполняется команда CALL, селектор которой указывает на дескриптор сегмента типа TSS.
    2. В TSS текущей задачи сохраняются значения регистров процессора. На текущий сегмент TSS указывает регистр процессора TR, содержащий селектор сегмента.
    3. В TR загружается селектор сегмента TSS задачи, на которую переключается процессор.
    4. Из нового TSS в регистр LDTR переносится значение селектора таблицы LDT в таблице GDT задачи.
    5. Восстанавливаются значения регистров процессора (из соответствующих полей нового сегмента TSS).
    6. В поле селектора возврата заносится селектор сегмента TSS снимаемой с выполнения задачи для организации возврата к прерванной задаче в будущем.

**Выводы**

В данной практической работе были изучены особенности защищенного режима процессора. Получены практические навыки по программированию переключения процессора из реального в защищенный режим и обратно.

# Приложение 1. Листинг кода

;---Define----------------------------------------------------------------------

%define DATA\_SEG 0x0010

;---Initialized data------------------------------------------------------------

hello\_world\_message dw 12

db 'Hello World!'

ticks\_message dw 20

db 'System timer ticks: '

scancode\_message dw 20

db 'Keyboard scan code: '

task1\_message dw 6

db "Task 1"

task2\_message dw 6

db "Task 2"

task3\_message dw 6

db "Task 3"

;---Include---------------------------------------------------------------------

%include "kernel/video.asm"

%include "kernel/idt.asm"

%include "kernel/isr.asm"

%include "kernel/tasking.asm"

;---Code------------------------------------------------------------------------

Kernel:

lidt [IDTR] ; Load our IDTR

mov al, 0x80 ; OCW1: Unmask all interrupts at master PIC.

out PIC1\_DATA, al

mov al, 0x80 ; OCW1: Unmask all interrupts at master PIC.

out PIC2\_DATA, al

; Set all segments registers to DATA\_SEG.

mov ax, DATA\_SEG

mov ds, ax

mov es, ax

mov fs, ax

mov gs, ax

mov ss, ax

; Clear the screen.

mov rax, 0x0020002000200020 ; Set background color to black (0) and

; character to blank space (20).

call Fill\_screen

; Print "Hello World!" at the upper right corner.

mov ah, 0x1E

mov r8, 69

mov r9, 1

mov rsi, hello\_world\_message

call Print

; Initialize general stack allocation to the current rsp value.

mov [stack\_allocation], rsp

; Create three tasks.

mov rsi, Task1

call Create\_task

mov rsi, Task2

call Create\_task

mov rsi, Task3

call Create\_task

; Set active the first task slot

mov qword [active\_task\_slot], 0

; Task 1: We print system timer ticks and keyboard scan code.

Task1:

mov ah, (VGA\_COLOR\_DARK\_GREY << 4) | VGA\_COLOR\_CYAN

mov r8, 1

mov r9, 2

mov rsi, task1\_message

Call Print

mov r8, 1

mov r9, 3

mov rsi, ticks\_message

Call Print

mov r8, 1

mov r9, 4

mov rsi, scancode\_message

Call Print

.loop:

; Print system timer ticks.

mov r8, 21

mov r9, 3

mov r10, [systimer\_ticks]

call Print\_hex

; Print keyboard scan code.

mov r8, 21

mov r9, 4

mov r10, [keyboard\_scancode]

call Print\_hex

jmp Task1.loop

; Task 2: We set r10 to 0 and we increase it by one in a loop.

Task2:

mov ah, (VGA\_COLOR\_GREEN << 4) | VGA\_COLOR\_WHITE

mov r8, 1

mov r9, 6

mov rsi, task2\_message

Call Print

mov r8, 1

mov r9, 7

mov r10, 0

.loop:

inc r10

; Print number of ticks

Call Print\_hex

jmp Task2.loop

; Task 3: We set r10 to 0xFFFFFFFFFFFFFFFF and we decrease it by one in a loop.

Task3:

mov ah, (VGA\_COLOR\_MAGENTA << 4) | VGA\_COLOR\_WHITE

mov r8, 1

mov r9, 9

mov rsi, task3\_message

Call Print

mov r8, 1

mov r9, 10

mov r10, 0xFFFFFFFFFFFFFFFF

.loop:

dec r10

; Print number of ticks

Call Print\_hex

jmp Task3.loop