Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Архитерктура вычислительных систем»

ОТЧЕТ

к практической работе №2

на тему:

«МУЛЬТИЗАДАЧНОСТЬ В ЗАЩИЩЕННОМ РЕЖИМЕ»

БГУИР 1-40 04 01

Выполнил студент группы 253502

Альховик Данила Игоревич

_____

(дата, подпись студента)

Проверил ассистент кафедры информатики

Калиновская Анастасия

Александровна

_____

(дата, подпись преподавателя)

Минск 2024

Цель работы:

Изучить принципы и средства реализации мультизадачности в защищенном режиме процессора. 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

Ход работы: На рисунке 1 представлено значение в регистрах перед выполнением программы.

Листинг 1 – Исходный код программы задания

gtd.h

```
#include "../lib/stdint.h"

#define GDT_STD_SELECTOR_NULL       0
#define GDT_STD_SELECTOR_KERNEL_CS  1
#define GDT_STD_SELECTOR_KERNEL_DS  2
#define GDT_STD_SELECTOR_USR_CS     3
#define GDT_STD_SELECTOR_USR_DS     4

struct gdt_entry
{
    u16 limit_low;
    u16 base_low;
    u8 base_middle;
    u8 access;
    u8 flags;
    u8 base_high;
} __attribute__((packed));

struct gdt_ptr {
    u16 limit;
    u32 base;
} __attribute__((packed));

typedef struct {
    u32 prev_tss;
    u32 esp0;
    u32 ss0;
    u32 esp1;
    u32 ss1;
    u32 esp2;
    u32 ss2;
    u32 cr3;
    u32 eip;
    u32 eflags;
    u32 eax;
    u32 ecx;
    u32 edx;
```

```
    u32 ebx;
    u32 esp;
    u32 ebp;
    u32 esi;
    u32 edi;
    u32 es;
    u32 cs;
    u32 ss;
    u32 ds;
    u32 fs;
    u32 gs;
    u32 ldt;
    u16 trap;
    u16 iomap_base;
} __attribute__((packed)) tss_entry_t;

void setupGdt();
void destroyGdt();
void setGdtEntry(u32 id, u32 base, u32 limit, u8 access, u8 flags);

#define GDT_NULL_CS     (GDT_STD_SELECTOR_NULL * sizeof(struct gdt_entry))
#define GDT_KERNEL_CS   (GDT_STD_SELECTOR_KERNEL_CS * sizeof(struct
gdt_entry))
#define GDT_KERNEL_DS   (GDT_STD_SELECTOR_KERNEL_DS * sizeof(struct
gdt_entry))
#define GDT_USR_CS      (GDT_STD_SELECTOR_USR_CS * sizeof(struct gdt_entry))
#define GDT_USR_DS      (GDT_STD_SELECTOR_USR_DS * sizeof(struct gdt_entry))
```

gtd.c

```
#include "gdt.h"

extern void load_gdt(u32);

struct gdt_entry gdt[6];
struct gdt_ptr gp;

tss_entry_t tss1;

void setGdtEntry(u32 id, u32 base, u32 limit, u8 access, u8 flags)
{
    gdt[id].base_low = (base & 0xFFFF);
    gdt[id].base_middle = (base >> 16) & 0xFF;
    gdt[id].base_high = (base >> 24) & 0xFF;

    gdt[id].limit_low = (limit & 0xFFFF);
    gdt[id].flags = (limit >> 16) & 0x0F;

    gdt[id].flags |= (flags & 0xF0);
    gdt[id].access = access;
}

void setupGdt()
{
    gp.limit = (sizeof(struct gdt_entry) * 5) - 1;
```

```
    gp.base = (u32)&gdt;

    setGdtEntry(GDT_STD_SELECTOR_NULL, 0, 0, 0, 0); // Null segment
    setGdtEntry(GDT_STD_SELECTOR_KERNEL_CS, 0, 0xFFFFFFFF, 0x9A, 0xCF); //
System code segment
    setGdtEntry(GDT_STD_SELECTOR_KERNEL_DS, 0, 0xFFFFFFFF, 0x92, 0xCF); //
System data segment
    setGdtEntry(GDT_STD_SELECTOR_USR_CS, 0, 0xFFFFFFFF, 0xFA, 0xCF); // User
mode code segment
    setGdtEntry(GDT_STD_SELECTOR_USR_DS, 0, 0xFFFFFFFF, 0xF2, 0xCF); // User
mode data segment
    setGdtEntry(5, (u32)&tss1, sizeof(tss1), 0x89, 0x40); // TSS
    load_gdt((u32)&gp);
}
```

kernel.c

```
#include "vga/vga.h"
#include "gdt/gdt.h"
#include <stdint.h>

extern void disable_interrupts(void);
extern void reload_segments(void);
extern void enter_protected_mode(void);
extern void leave_protected_mode(void);

tss_entry_t tss4, tss5, tss6;

void set_tss(tss_entry_t* tss, u32 eip, u32 esp) {
    tss->esp0 = esp;
    tss->ss0 = 0x10;
    tss->eip = eip;
    tss->esp = esp;
    tss->cs = 0x08;
    tss->ss = tss->ds = tss->es = tss->fs = tss->gs = 0x10;
    tss->iomap_base = sizeof(tss_entry_t);
}

void task1(void) {
    for (int i = 0; i < 2; i++) {
        print("Task 1 - Part ");
        print_int(i);
        print("\n");
        for (volatile int j = 0; j < 1000000; j++); // Пустой цикл
    }
    print("Task 1 completed\n");
}

void task2(void) {
    for (int i = 0; i < 2; i++) {
        print("Task 2 - Part ");
        print_int(i);
        print("\n");
        for (volatile int j = 0; j < 1000000; j++); // Пустой цикл
```

```
    }
    print("Task 2 completed\n");
}

void task3(void) {
    for (int i = 0; i < 2; i++) {
        print("Task 3 - Part ");
        print_int(i);
        print("\n");
        for (volatile int j = 0; j < 1000000; j++); // Пустой цикл
    }
    print("Task 3 completed\n");
}

void setup(void) {
    disable_interrupts();
    setupGdt();
    reload_segments();
    enter_protected_mode();
}

void leave(void) {
    disable_interrupts();
    print("Disabling interrupts\n");
    leave_protected_mode();
    print("Leaving protected mode\n");
}

void kmain(void) {
    clear();
    print("Welcome to the real mode!\n");
    setup();
    print("You're in a protected mode now\n");

    // Настройка TSS для задач
    set_tss(&tss4, (u32)task1, 0x9FC00);
    set_tss(&tss5, (u32)task2, 0x9F800);
    set_tss(&tss6, (u32)task3, 0x9F400);

    // Запуск задач
    asm volatile ("ltr %%ax" : : "a"(0x28)); // Загрузка TSS для task1
    asm volatile ("jmp $0x20, $0"); // Переход к task1

    leave();
    print("You're back in real mode\n");

    while (1) {
        // Бесконечный цикл для предотвращения завершения программы
    }
}
```

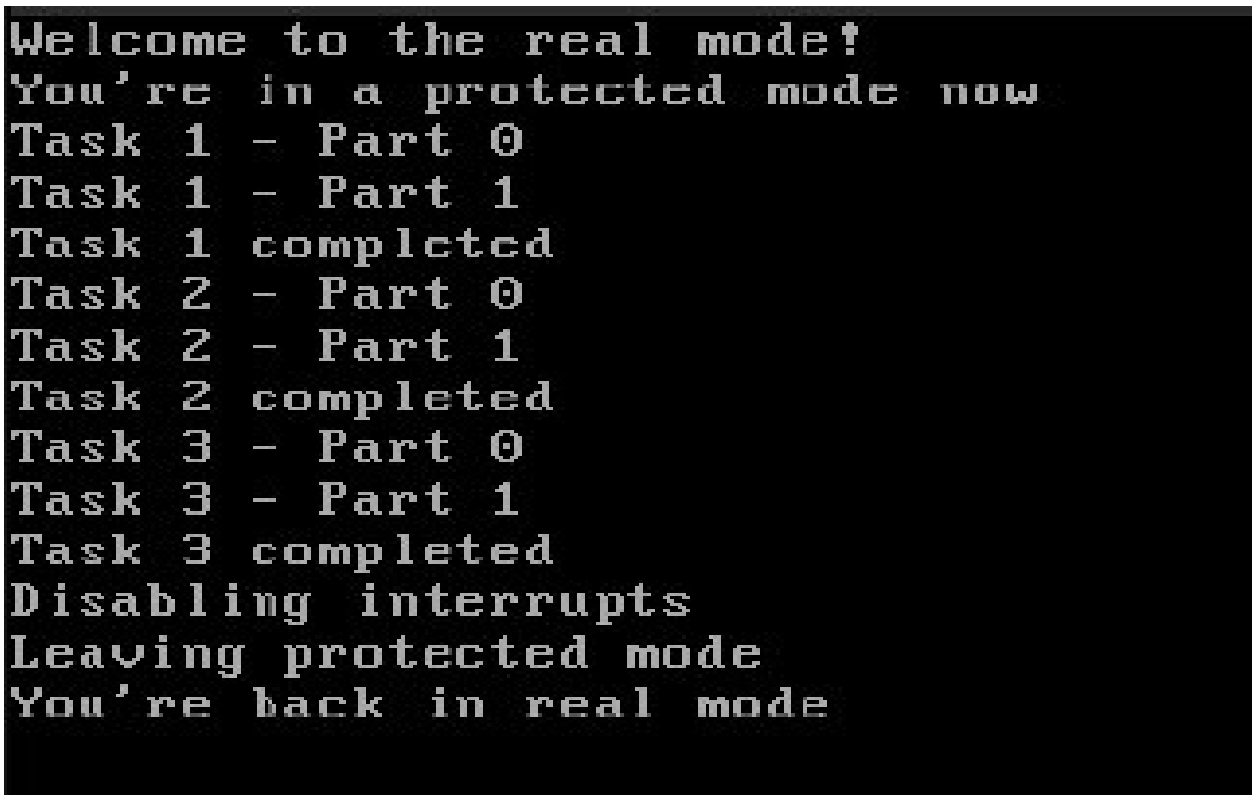

Рисунок 1 — Результат выполнения программы

Выводы: произведен запуск программы, которая запустила защищенный режим процессора из реального выполнила задачи с использованием переключений между ними, а затем снова вернула процессор в реальный режим.