МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЖУРНАЛ
ПО ВЫЧИСЛИТЕЛЬНОЙ ПРАКТИКЕ

Наименование практики *вычислительная*

Студенты:

Шубин Григорий
Артамонов Олег
Поляков Андрей

Факультет № 8 курс 2 группа 8

Практика с 28.06.21 по 12.07.21

ИНСТРУКЦИЯ

о заполнении журнала по вычислительной практике

Журнал по вычислительной практике студентов имеет единую форму для всех видов практик.

Задание в журнал вписывается руководителем практики от института в первые три – пять дней пребывания студентов на практике в соответствии с тематикой, утверждённой на кафедре до начала практики. Журнал по вычислительной практике является основным документом для текущего и итогового контроля выполнения заданий, требований инструкции и программы практики.

Табель прохождения практики, задание, а также технический отчёт выполняются каждым студентом самостоятельно.

Журнал заполняется студентом непрерывно в процессе прохождения всей практики и регулярно представляется для просмотра руководителям практики. Все их замечания подлежат немедленному выполнению.

В разделе «Табель прохождения практики» ежедневно должно быть указано, на каких рабочих местах и в качестве кого работал студент. Эти записи проверяются и заверяются цеховыми руководителями практики, в том числе мастерами и бригадирами. График прохождения практики заполняется в соответствии с графиком распределения студентов по рабочим местам практики, утверждённым руководителем предприятия.

В разделе «Рационализаторские предложения» должно быть приведено содержание поданных в цехе рационализаторских предложений со всеми необходимыми расчётами и эскизами. Рационализаторские предложения подаются индивидуально и коллективно.

Выполнение студентом задания по общественно-политической практике заносятся в раздел «Общественно-политическая практика». Выполнение работы по оказанию практической помощи предприятию (участие в выполнении спецзаданий, работа сверхурочно и т.п.) заносятся в раздел журнала «Работа в помощь предприятию» с последующим письменным подтверждением записанной работы соответствующими цеховыми руководителями.

Раздел «Технический отчёт по практике» должен быть заполнен особо тщательно. Записи необходимо делать чернилами в сжатой, но вместе с тем чёткой и ясной форме и технически грамотно. Студент обязан ежедневно подробно излагать содержание работы, выполняемой за каждый день. Содержание этого раздела должно отвечать тем конкретным требованиям, которые предъявляются к техническому отчёту заданием и программой практики. Технический отчёт должен показать умение студента критически оценивать работу данного производственного участка и отразить, в какой степени студент способен применить теоретические знания для решения конкретных производственных задач.

Иллюстративный и другие материалы, использованные студентом в других разделах журнала, в техническом отчёте не должны повторяться, следует ограничиваться лишь ссылкой на него. Участие студентов в производственно-технической конференции, выступление с докладами, рационализаторские предложения и т.п. должны заноситься на свободные страницы журнала.

Примечание. Синьки, кальки и другие дополнения к журналу могут быть сделаны только с разрешения администрации предприятия и должны подшиваться в конце журнала.

Руководители практики от института обязаны следить затем, чтобы каждый цеховой руководитель практики перед уходом студентов из данного цеха в другой цех вписывал в журнал студента отзывы об их работе в цехе.

Текущий контроль работы студентов осуществляется руководители практики от института и цеховыми руководителями практики заводов. Все замечания студентам руководители делают в письменном виде на страницах журнала, ставя при этом свою подпись и дату проверки.

Результаты защиты технического отчёта заносятся в протокол и одновременно заносятся в ведомость и зачётную книжку студента.

Примечание. Нумерация чистых страниц журнала проставляется каждым студентом в своём журнале до начала практики.

С инструкцией о заполнении журнала ознакомился:

«12» июля 2021г.

Студент *Шубин Григорий*

_____

(подпись)

Студент *Артамонов Олег*

_____

(подпись)

Студент *Поляков Андрей*

_____

(подпись)

ЗАДАНИЕ

кафедры 806 по вычислительной практике

1. **Создать загрузочный образ миниядра MiniOS.**
2. **Изучить в нём механизм прерываний.**
3. Составить алгоритм.
4. Реализовать механизм выделения статической памяти.
5. Тестирование алгоритма.
6. Список используемой литературы.
7. Выводы.

Руководитель практики
от института

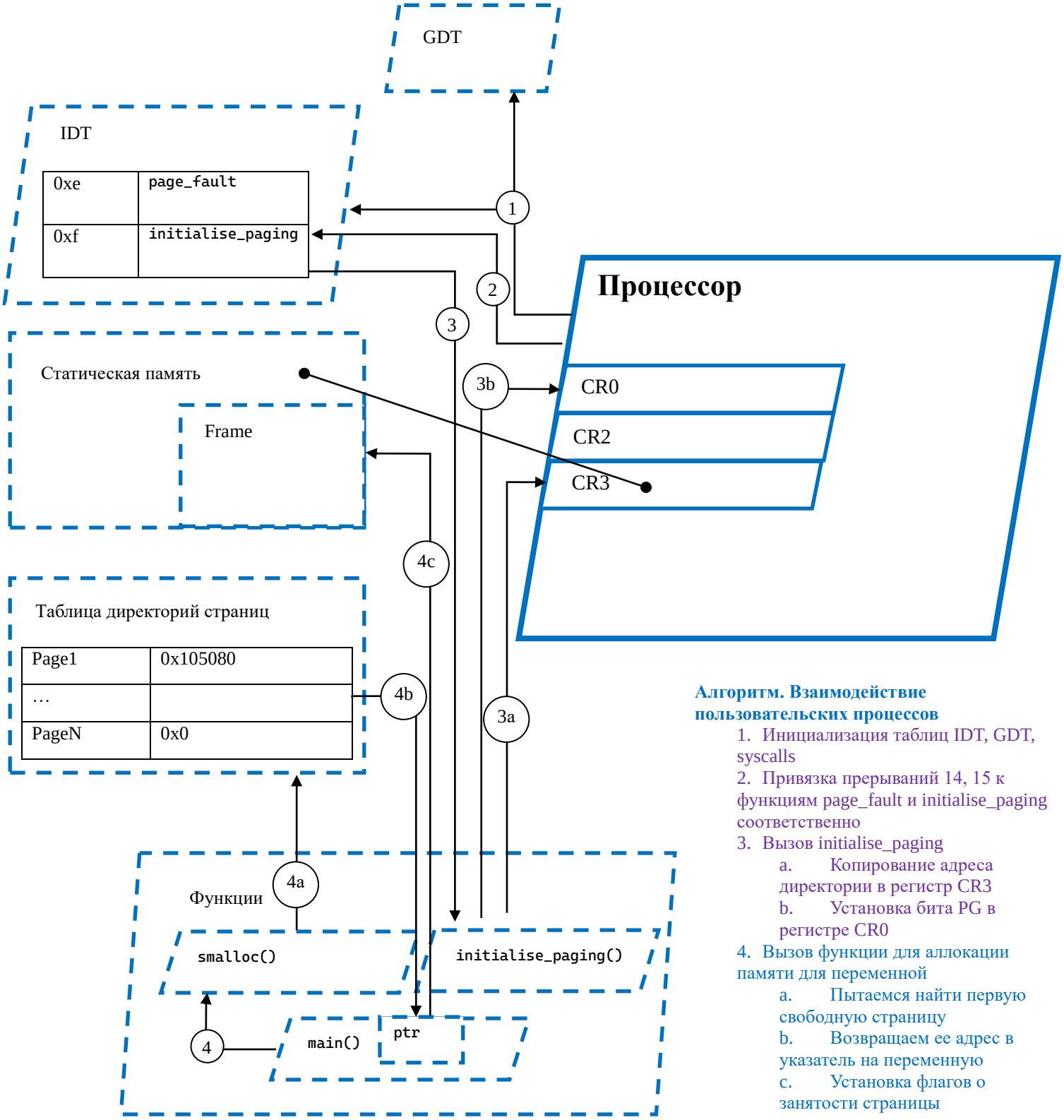
«12» июля 2021 г. **Подпись**

ПРОТОКОЛ
ЗАЩИТЫ ТЕХНИЧЕСКОГО ОТЧЁТА

по *вычислительной практике*
студентами 1. *Шубин Григорий*
2. *Артамонов Олег*
3. *Поляков Андрей*

Слушали: Отчёт практиканта	Постановили: Считать практику выполненной и защищённой на
1. Создать загрузочный образ миниядра MiniOS.	Оценка 1. _____ Оценка 2. _____ Оценка 3. _____
2. Изучить в нём механизм прерываний	Оценка 1. _____ Оценка 2. _____ Оценка 3. _____
3. Составить алгоритм.	Оценка 1. _____ Оценка 2. _____ Оценка 3. _____
4.Реализовать механизм выделения статической памяти.	Оценка 1. _____ Оценка 2. _____ Оценка 3. _____
5. Тестирование алгоритма.	Оценка 1. _____ Оценка 2. _____ Оценка 3. _____
6. Список используемой литературы.	Оценка 1. _____ Оценка 2. _____ Оценка 3. _____
7. Выводы.	Оценка 1. _____ Оценка 2. _____ Оценка 3. _____
	Общая оценка - _____

Руководитель: Семенов А. С. _____
Дата: 12.07.2021



Алгоритм. Взаимодействие пользовательских процессов

1. Инициализация таблиц IDT, GDT, syscalls
2. Привязка прерываний 14, 15 к функциям page_fault и initialise_paging соответственно
3. Вызов initialise_paging
   a. Копирование адреса директории в регистр CR3
   b. Установка бита PG в регистре CR0
4. Вызов функции для аллокации памяти для переменной
   a. Пытаемся найти первую свободную страницу
   b. Возвращаем ее адрес в указатель на переменную
   c. Установка флагов о занятости страницы

№	Команда	Комментарии
0	initialise_paging()	Инициализация страничной памяти.

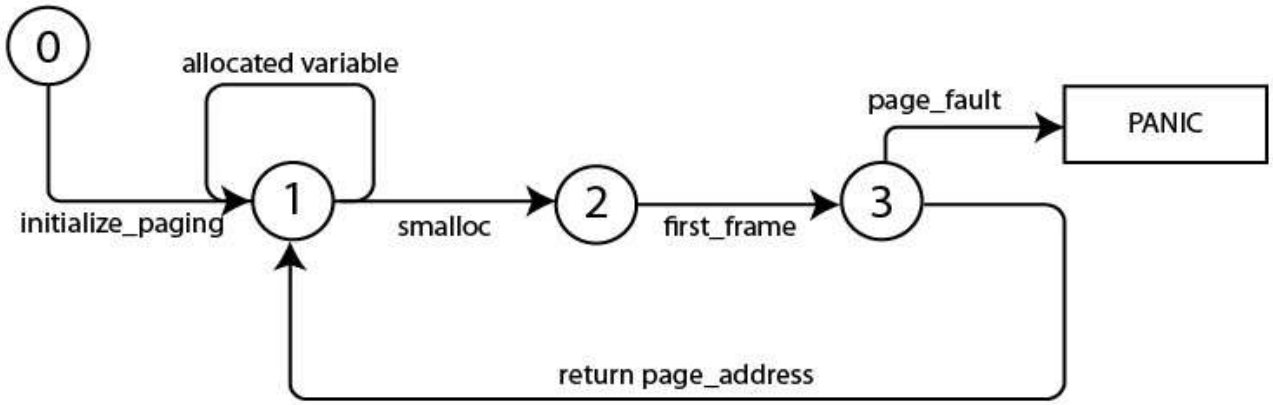
1	u32int *ptr = (u32int*)0xA0000000;
	
	u32int do_page_fault = *ptr;
		Обращение к ячейке памяти.

2	аппаратный вызов	Обращение процессора к таблице прерываний.
3	page_fault(register_t regs)	Вызов обработчика ошибки.
4	asm volatile("mov %%cr2, %0" : "=r" (faulting_address))	Считывание адреса с регистра для обработки ошибки.

5	monitor_write("Page fault! (");
	
	…….
	
	monitor_write_hex(faulting_address);
		Обработка ошибки выделения памяти.

6	smalloc()	Выделение памяти для переменной типа u32int в статической памяти

Такты системы



(0,cr2=0x0,cr3=0x0,cr4=0x0,IDT<0,…,0>,current_directory<0,…,0>)
⊢ init_idt
(1,cr2=0x0,cr3=0x0,cr4=0x0,IDT<0,…,0,page_fault,0,...,0,page_init,page_find>,current_directory<0,…,0>)
⊢ initialise_paging
(2,cr2=0x0,cr3=0x0,cr4=0x0,IDT<0,…,0,page_fault,0,...,0,page_init,page_find>,current_directory<page1,...,pageN>)
⊢ switch_page_directory
(3,cr2=0x0,cr3=current_directory,cr4=0x0,IDT<0,…,0,page_fault,0,...,0,page_init,page_find>,current_directory<page1,...,pageN>)
⊢ page_fault
(4,cr2=0x1000001,cr3=current_directory,cr4=0x0,IDT<0,…,0,page_fault,0,...,0,page_init,page_find>,current_directory<page1,...,pageN>)

Управление статической памятью в Linux

В пространстве пользователя многие операции выделения памяти, в частности некоторые рассмотренные ранее примеры, могут быть выполнены с использованием стека, потому что априори известен размер выделяемой области памяти. В пространстве пользователя доступна такая роскошь, как очень большой и динамически увеличивающийся стек задачи, однако в режиме ядра такой роскоши нет — стек ядра маленький и фиксирован по размеру. Когда процессу выделяется небольшой и фиксированный по размеру стек, то затраты памяти уменьшаются и ядру нет необходимости выполнять дополнительные функции по управлению памятью.

Размер стека зависит как от аппаратной платформы, так и от конфигурационных параметров, которые были указаны на этапе компиляции. Исторически размер стека ядра был равен двум страницам памяти для каждого процесса. Это соответствует 8 Кбайт для 32-разрядных аппаратных платформ и 16 Кбайт для 64-разрядных аппаратных платформ.

Стеки обработчиков прерываний. Стеки прерываний представляют собой один стек на каждый процессор, которые используются для обработки прерываний. При такой конфигурации обработчики прерываний больше не используют стеки ядра тех процессов, которые этими обработчиками прерываются. Вместо этого

они используют свои собственные стеки. Это требует только одну страницу памяти на процессор.

Стек ядра занимает одну или две страницы памяти, в зависимости от конфигурации, которая выполняется перед компиляцией ядра. Следовательно, размер стека ядра может иметь диапазон от 4 до 16 Кбайт. Исторически обработчики прерываний совместно использовали стек прерванного ими процесса. При появлении стеков ядра размером в одну страницу памяти обработчикам прерываний были назначены свои стеки. В любом случае неограниченная рекурсия и использование функций вроде alloca() явно не допустимы.

Использование ООП

Автоматизация доступа к памяти в C++ увеличивает затраты памяти и замедляет работу программ. Многие детали поведения кода стандартом C++ не специфицированы, что ухудшает переносимость и может являться причиной трудно обнаруживаемых ошибок.

В низкоуровневом программировании значительная часть новых возможностей C++ оказывается неприменимой из-за увеличения накладных расходов: виртуальные функции требуют динамического вычисления реального адреса (RVA), шаблоны приводят к раздуванию кода и ухудшению возможностей оптимизации, библиотека времени исполнения (RTL) очень велика, а отказ от неё лишает большинства возможностей C++ (хотя бы из-за недоступности операций new/delete). В результате программисту придётся ограничиться функционалом, унаследованным от Си, что делает бессмысленным применение C++.

Применение ООП возможно, но не несет за собой практически никаких преимуществ, что исключает востребованность данного подхода при разработке компонентов операционных систем.

4. Реализация процедуры:

smem.h

```
#ifndef SMEM_H
#define SMEM_H
#include "common.h"
u32int kmalloc_int(u32int sz, int align, u32int *phys);
u32int kmalloc_a(u32int sz);
u32int kmalloc_p(u32int sz, u32int *phys);
u32int kmalloc_ap(u32int sz, u32int *phys);
u32int kmalloc(u32int sz);
#endif // SMEM_H
```

smem.c

```
#include "smem.h"
extern u32int end;
u32int placement_address = (u32int)&end;
u32int kmalloc_int(u32int sz, int align, u32int *phys)
{
        if (align == 1 && (placement_address & 0xFFFFF000) )
    {
    placement_address &= 0xFFFFF000;
    placement_address += 0x1000;
    }
    if (phys)
    {
    *phys = placement_address;
    }
    u32int tmp = placement_address;
    placement_address += sz;
    return tmp;
}
```

```
u32int kmalloc_a(u32int sz)



{
	return kmalloc_int(sz, 1, 0);
}
u32int kmalloc_p(u32int sz, u32int *phys)
{
	return kmalloc_int(sz, 0, phys);
}
u32int kmalloc_ap(u32int sz, u32int *phys)
{
	return kmalloc_int(sz, 1, phys);
}
u32int kmalloc(u32int sz)
{
	return kmalloc_int(sz, 0, 0);
}
```

paging.h

```
#ifndef PAGING_H
#define PAGING_H
#include "common.h"
#include "isr.h"
typedef struct page
{
	u32int present : 1;    // Page present in memory
	u32int rw      : 1;    // Read-only if clear, readwrite if set
	u32int user    : 1;    // Supervisor level only if clear
	u32int accessed    : 1;    // Has the page been accessed since last refresh?
	u32int dirty    : 1;    // Has the page been written to since last refresh?
	u32int unused          : 7;    // Amalgamation of unused and reserved bits
```

```
        u32int frame    : 20;   // Frame address (shifted right 12 bits)



} page_t;

typedef struct page_table

{

        page_t pages[1024];

} page_table_t;

typedef struct page_directory

{

        page_table_t *tables[1024];

        u32int tablesPhysical[1024];

        u32int physicalAddr;

} page_directory_t;

void initialise_paging();

void switch_page_directory(page_directory_t *new);

page_t *get_page(u32int address, int make, page_directory_t *dir);

void page_fault(registers_t regs);

#endif
```

paging.c

```
#include "paging.h"

#include "kheap.h"

// The kernel's page directory

page_directory_t *kernel_directory=0;

// The current page directory;

page_directory_t *current_directory=0;

// A bitset of frames - used or free.

u32int *frames;

u32int nframes;

// Defined in kheap.c

extern u32int placement_address;
```

```
// Macros used in the bitset algorithms.


#define INDEX_FROM_BIT(a) (a/(8*4))
#define OFFSET_FROM_BIT(a) (a%(8*4))
// Static function to set a bit in the frames bitset
static void set_frame(u32int frame_addr)
{
	u32int frame = frame_addr/0x1000;
	u32int idx = INDEX_FROM_BIT(frame);
	u32int off = OFFSET_FROM_BIT(frame);
	frames[idx] |= (0x1 << off);
}
// Static function to clear a bit in the frames bitset
static void clear_frame(u32int frame_addr)
{
	u32int frame = frame_addr/0x1000;
	u32int idx = INDEX_FROM_BIT(frame);
	u32int off = OFFSET_FROM_BIT(frame);
	frames[idx] &= ~(0x1 << off);
}
// Static function to test if a bit is set.
static u32int test_frame(u32int frame_addr)
{
	u32int frame = frame_addr/0x1000;
	u32int idx = INDEX_FROM_BIT(frame);
	u32int off = OFFSET_FROM_BIT(frame);
	return (frames[idx] & (0x1 << off));
}
// Static function to find the first free frame.
static u32int first_frame()
```

```
{
	u32int i, j;

	for (i = 0; i < INDEX_FROM_BIT(nframes); i++)
	{
	if (frames[i] != 0xFFFFFFFF) // nothing free, exit early.
	{
	// at least one bit is free here.
	for (j = 0; j < 32; j++)
	{
		u32int toTest = 0x1 << j;
		if ( !(frames[i]&toTest) )
		{
		return i*4*8+j;
		}
	}
	}
	}
}
// Function to allocate a frame.
void alloc_frame(page_t *page, int is_kernel, int is_writeable)
{
	if (page->frame != 0)
	{
	return;
	}
	else
	{
	u32int idx = first_frame();
	if (idx == (u32int)-1)
	{
	// PANIC! no free frames!!
```

```
        }
        set_frame(idx*0x1000);

        page->present = 1;
        page->rw = (is_writeable)?1:0;
        page->user = (is_kernel)?0:1;
        page->frame = idx;
        }
}
// Function to deallocate a frame.
void free_frame(page_t *page)
{
        u32int frame;
        if (!(frame=page->frame))
        {
        return;
        }
        else
        {
        clear_frame(frame);
        page->frame = 0x0;
        }
}
void initialise_paging()
{
        // The size of physical memory. For the moment we
        // assume it is 16MB big.
        u32int mem_end_page = 0x1000000;

        nframes = mem_end_page / 0x1000;
        frames = (u32int*)kmalloc(INDEX_FROM_BIT(nframes));
        memset(frames, 0, INDEX_FROM_BIT(nframes));
```

```
	// Let's make a page directory.

	kernel_directory = (page_directory_t*)kmalloc_a(sizeof(page_directory_t));
	current_directory = kernel_directory;
	// We need to identity map (phys addr = virt addr) from
	// 0x0 to the end of used memory, so we can access this
	// transparently, as if paging wasn't enabled.
	// NOTE that we use a while loop here deliberately.
	// inside the loop body we actually change placement_address
	// by calling kmalloc(). A while loop causes this to be
	// computed on-the-fly rather than once at the start.
	int i = 0;
	while (i < placement_address)
	{
	// Kernel code is readable but not writeable from userspace.
	alloc_frame( get_page(i, 1, kernel_directory), 0, 0);
	i += 0x1000;
	}
	// Before we enable paging, we must register our page fault handler.
	register_interrupt_handler(14, page_fault);
	// Now, enable paging!
	switch_page_directory(kernel_directory);
}
void switch_page_directory(page_directory_t *dir)
{
	current_directory = dir;
	asm volatile("mov %0, %%cr3":: "r"(&dir->tablesPhysical));
	u32int cr0;
	asm volatile("mov %%cr0, %0": "=r"(cr0));
	cr0 |= 0x80000000; // Enable paging!
```

```
        asm volatile("mov %0, %%cr0":: "r"(cr0));



}
page_t *get_page(u32int address, int make, page_directory_t *dir)
{
        // Turn the address into an index.
        address /= 0x1000;
        // Find the page table containing this address.
        u32int table_idx = address / 1024;
        if (dir->tables[table_idx]) // If this table is already assigned
        {
        return &dir->tables[table_idx]->pages[address%1024];
        }
        else if(make)
        {
        u32int tmp;
        dir->tables[table_idx] = (page_table_t*)kmalloc_ap(sizeof(page_table_t), &tmp);
        dir->tablesPhysical[table_idx] = tmp | 0x7; // PRESENT, RW, US.
        return &dir->tables[table_idx]->pages[address%1024];
        }
        else
        {
        return 0;
        }
}

void page_fault(registers_t regs)
{
        // A page fault has occurred.
        // The faulting address is stored in the CR2 register.
        u32int faulting_address;
```

```
        asm volatile("mov %%cr2, %0" : "=r" (faulting_address));



        // The error code gives us details of what happened.
        int present    = !(regs.err_code & 0x1); // Page not present
        int rw = regs.err_code & 0x2;           // Write operation?
        int us = regs.err_code & 0x4;           // Processor was in user-mode?
        int reserved = regs.err_code & 0x8;   // Overwritten CPU-reserved bits of page entry?
        int id = regs.err_code & 0x10;          // Caused by an instruction fetch?
        // Output an error message.
        monitor_write("Page fault! ( ");
        if (present) {monitor_write("present ");}
        if (rw) {monitor_write("read-only ");}
        if (us) {monitor_write("user-mode ");}
        if (reserved) {monitor_write("reserved ");}
        monitor_write(") at 0x");
        monitor_write_hex(faulting_address);
        monitor_write("\n");
        PANIC("Page fault");
}

u32int* smalloc(){
        u32int * ptr = first_frame();
        if (!ptr){
                asm volatile("int $0xe");
                return;
        }
        return ptr;
}
```

main.c

```
#include "monitor.h"
#include "multiboot.h"
```

```
#include "descriptor_tables.h"
#include "timer.h"
#include "paging.h"

void proc1(u32int* n) {
	monitor_write("Unit 1 accessing variable:\n");
	monitor_write_hex(*n);
	monitor_write("\n");
}
void proc2(u32int* n) {
	monitor_write("Unit 2 accessing variable:\n");
	monitor_write_hex(*n);
	monitor_write("\n");
}
int main(struct multiboot* mboot_ptr) {
	init_descriptor_tables();

	monitor_clear();

	init_interuptions();

	asm volatile("int $0xf");
	asm volatile("sti");
	    monitor_write("----------------\n");
	//Testing global variable allocation
	u32int* ptr = (u32int*)0xF0;
	ptr = 0xf;
	monitor_write("Initializing global variable with value of: ");
	monitor_write_hex(ptr);
	monitor_write(", at adress:");
	monitor_write_hex(&ptr);
```

```
	monitor_write("\n");
	proc1(&ptr);
	proc2(&ptr);

	monitor_write("----------------\n");

	//Test of choosing free pages
	monitor_write("Test of choosing free pages. \n Trying to allocate 2 variables:\n");
	u32int* ptr2 = smalloc();
	monitor_write("First allocated at: ");
	monitor_write_hex(&ptr2);
	monitor_write("\n");

	u32int* ptr3 = smalloc();
	monitor_write("Second allocated at: ");
	monitor_write_hex(&ptr3);
	monitor_write("\n");

	/// init_timer(10);

	//u32int *ptr = (u32int*)0xA0000000;

	//u32int do_page_fault = *ptr;

	return 0;
}
```

5. Тестирование программы.

```
Hello, paging world!
recieved interrupt: 3
recieved interrupt: 4
recieved interrupt: 14
Page fault! ( present ) at 0x0xa0000000
PANIC(Page fault) at paging.c:201
```

```
Init interuption 14 done
Init interuption 15 done
recieved interrupt: 15
Paging initialized
---------------
Initializing global variable with value of: 0xf, at adress:0x67e1c
Unit 1 accessing variable:
0xf
Unit 2 accessing variable:
0xf
---------------
Test of choosing free pages.
 Trying to allocate 2 variables:
First allocated at: 0x67e18
Second allocated at: 0x67e14
Done!_
```

СПИСОК ЛИТЕРАТУРЫ

1. Документация MiniOS
2. Проектирование сетевых операционных систем/А.С.Семёнов — Москва: Вузовская книга, 2008
3. Руководство по созданию простой UNIX-подобной ОС [Электронный ресурс] URL:http://rus-linux.net/MyLDP/kernel/toyos/sozdaem-unix-like-os.html
4. Chapter 4.The GDT and IDT [Электронный ресурс]: http://www.jamesmolloy.co.uk URL:http://www.jamesmolloy.co.uk/tutorial_html/4.-The%20GDT%20and%20IDT.html
5. Chapter 5.IRQs and the PIT [Электронный ресурс]: http://www.jamesmolloy.co.uk URL:http://www.jamesmolloy.co.uk/tutorial_html/5.-IRQs%20and%20the%20PIT.html
6. MOS [Электронный ресурс]: mysticos.combuster.nl URL:https://mysticos.combuster.nl/?p=downloads
7. Chapter 8.The VFS and the initrd [Электронный ресурс]: http://www.jamesmolloy.co.uk URL:http://www.jamesmolloy.co.uk/tutorial_html/8.-The%20VFS%20and%20the%20initrd.html
8. Chapter 9.Multitasking [Электронный ресурс]: http://www.jamesmolloy.co.uk URL:http://www.jamesmolloy.co.uk/tutorial_html/9.-Multitasking.html
9. James Molloy's Tutorial Known Bugs [Электронный ресурс]: https://wiki.osdev.org URL:https://wiki.osdev.org/James_Molloy%27s_Tutorial_Known_Bugs
10. Язык Ада в проектировании систем/Р.Бар — Москва: Мир, 1988
11. X86 Assembly Language and C Fundamentals/J.Cavanagh — Лондон: CRC Press, 2013
12. OSDev.org [Электронный ресурс] URL:https://forum.osdev.org/
13. github.com [Электронный ресурс] URL:https://github.com/sukwon0709/osdev
14. Операционные системы/Э.Танненбаум, А.Вудхалл — Санкт-Петербург: Питер, 2007
15. Пособие по разработке макросов NASM
16. Interactive map of Linux kernel [Электронный ресурс]:www.makelinux.net URL:http://www.makelinux.net/kernel_map/
17. Wikipedia INT(x86 instruction) [Электронный ресурс] URL:https://en.wikipedia.org/wiki/INT_(x86_instruction)
18. Wikipedia Interrupt descriptor table [Электронный ресурс] URL:https://en.wikipedia.org/wiki/Interrupt_descriptor_table

ВЫВОДЫ

1. Так как в Ubuntu 18.04 нет grub, но есть grub2, то необходимо использовать предыдущую версию, например 16.04, чтобы компиляция файлов ядра прошла успешно, потому что в этой сборке можно установить grub
2. Необходимо изменить makefile для корректного создания загрузочного образа
3. В процессе выполнения задания изучили механизм прерываний.
4. Каждую из сборок легко собрать, немного изменив первоначальный make файл.
5. Много полезной информации нашлось на иностранных ресурсах.