МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЖУРНАЛ
ПО ВЫЧИСЛИТЕЛЬНОЙ ПРАКТИКЕ

Наименование практики *вычислительная*

Студенты:

Хренникова Ангелина
Калининна Анастасия
Назарова Анастасия

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: Хренникова А. С.

_____
(подпись)

Студент Калинина А. В.

_____
(подпись)

Студент Назарова А. И.

_____
(подпись)

ЗАДАНИЕ

кафедры 806 по вычислительной практике

1. Создать загрузочный образ миниядра MiniOS.
2. Изучить в нём механизм прерываний.
3. Составить алгоритм управления виртуальной памятью.
4. Реализовать алгоритм управления виртуальной памятью.
5. Тестирование алгоритма.
6. Список используемой литературы.
7. Выводы.

Руководитель практики
от института

«12» июля 2021 г. **Подпись**

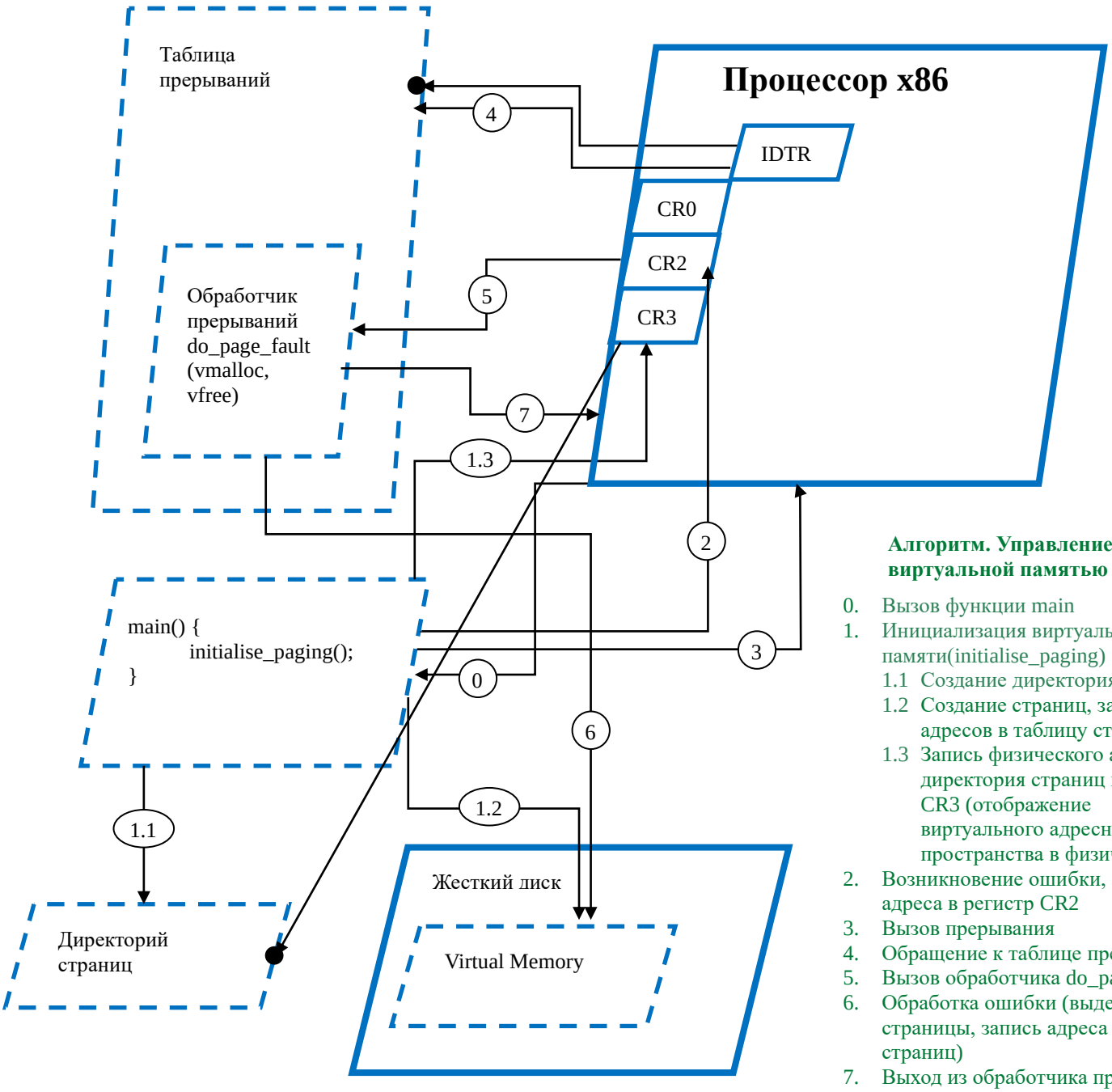
ПРОТОКОЛ

ЗАЩИТЫ ТЕХНИЧЕСКОГО ОТЧЁТА

по *вычислительной практике*
студентами 1. *Хренникова Ангелина*
2. *Калинина Анастасия*
3. *Назарова Анастасия*

Слушали:	Постановили:
Отчёт практиканта	Считать практику выполненной и защищённой на
1. Создать загрузочный образ мини ядра 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 алгоритм управления виртуальной памятью.	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. Управление виртуальной памятью

0. Вызов функции main
1. Инициализация виртуальной памяти(initialise_paging)
   - 1.1 Создание директория страниц
   - 1.2 Создание страниц, запись их адресов в таблицу страниц
   - 1.3 Запись физического адреса директория страниц в регистр CR3 (отображение виртуального адресного пространства в физическое)
2. Возникновение ошибки, запись ее адреса в регистр CR2
3. Вызов прерывания
4. Обращение к таблице прерываний
5. Вызов обработчика do_page_fault
6. Обработка ошибки (выделение страницы, запись адреса в таблицу страниц)
7. Выход из обработчика прерываний

Номер	Команда	Описание

0	int main() {
	…
	}
		Вызов функции main

1
(1.1-
1.3)
	initialise_paging() {
	…
	kernel_directory = (page_directory_t*)kmalloc_a(sizeof(page_directory_t));
	current_directory = kernel_directory;
	…
	for (i = KHEAP_START; i < KHEAP_START+KHEAP_INITIAL_SIZE; i += 0x1000)
	get_page(i, 1, kernel_directory);
	…
	switch_page_directory(kernel_directory);
	};
	
	void switch_page_directory(page_directory_t *dir)
	{
	current_directory = dir;
	asm volatile("mov %0, %%cr3":: "r"(&dir->tablesPhysical));
	}
		Инициализация виртуальной памяти(initialise_paging):
		Создание директория страниц;
		Создание страниц, запись их адресов в таблицу страниц;
		Запись физического адреса директория страниц в регистр CR3 (отображение виртуального адресного пространства в физическое);

2-3	u32int faulting_address; asm volatile("mov %%cr2, %0" : "=r" (faulting_address));	Возникновение ошибки, запись ее адреса в регистр CR2 и вызов прерывания
4	register_interrupt_handler(14, do_page_fault);	Обращение к таблице прерываний
5	do_page_fault()	Вызов обработчика do_page_fault

6	do_page_fault(registers_t regs) {
	…
	vmalloc(placement_address, current_directory);
	…
	}
		Обработка ошибки (выделение страницы, запись адреса в таблицу страниц)

7	PANIC("Page fault");	Выход из обработчика прерываний

4. Реализация процедуры:

```
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));
    memset(kernel_directory, 0, sizeof(page_directory_t));
    current_directory = kernel_directory;

    // Map some pages in the kernel heap area.
    // Here we call get_page but not alloc_frame. This causes page_table_t's
    // to be created where necessary. We can't allocate frames yet because they
    // they need to be identity mapped first below, and yet we can't increase
    // placement_address between identity mapping and enabling the heap!
    int i = 0;
    for (i = KHEAP_START; i < KHEAP_START+KHEAP_INITIAL_SIZE; i += 0x1000)
        get_page(i, 1, 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.
    // Allocate a lil' bit extra so the kernel heap can be
    // initialised properly.
    i = 0;
    while (i < placement_address+0x1000)
    {
        // Kernel code is readable but not writeable from userspace.
        alloc_frame( get_page(i, 1, kernel_directory), 0, 0);
        i += 0x1000;
    }

    // Now allocate those pages we mapped earlier.
    //for (i = KHEAP_START; i < KHEAP_START+KHEAP_INITIAL_SIZE; i += 0x1000)
    //    alloc_frame( get_page(i, 1, kernel_directory), 0, 0);

    // 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);

    // Initialise the kernel heap.
    //kheap = create_heap(KHEAP_START, KHEAP_START+KHEAP_INITIAL_SIZE, 0xCFFFF0
00, 0, 0);
}

void vmalloc(u32int placement_address, page_directory_t *current_directory) {
    alloc_frame( get_page(placement_address, 1, current_directory), 0, 0);
    placement_address += 0x1000;
}

void vfree(u32int placement_address, page_directory_t *current_directory) {
    free_frame(get_page(placement_address, 0, current_directory));
    placement_address -= 0x1000;
}
```

```
void do_page_fault(registers_t regs)
{
    u32int faulting_address;
    asm volatile("mov %%cr2, %0" : "=r" (faulting_address));

    monitor_write("The error handler runs!\n");

    if(faulting_address > 0x1000000) {
    //если адрес за пределами физ.памяти, то не пытаемся что-
тто сделать и отрубаем ядро
        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("Unrestorable 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");
    }

    //monitor_write("page fault detected ");
    monitor_write("Error at the address: ");
    monitor_write_hex(faulting_address);
    monitor_write("\n");

    //monitor_write("attempting to recover: ");
    monitor_write("Allocating one page!\n");
    monitor_write_hex(placement_address);
    monitor_write(" to ");
    monitor_write_hex(placement_address + 0x1000);
    monitor_write("\n");

    vmalloc(placement_address, current_directory);
    if (test_frame(placement_address, current_directory)) {
        vfree(placement_address, current_directory);
    }
}
```

5. Тестирование программы.

```
The error handler runs!
Error at the address: 0x10ffff
Allocating one page!
0x10d000 to 0x10e000
recieved interrupt: 14
The error handler runs!
Error at the address: 0x10ffff
Allocating one page!
0x10e000 to 0x10f000
recieved interrupt: 14
The error handler runs!
Error at the address: 0x10ffff
Allocating one page!
0x10f000 to 0x110000
recieved interrupt: 14
The error handler runs!
Error at the address: 0x110002
Allocating one page!
0x110000 to 0x111000
Appeal to the impossible address: 0xff10ffff
recieved interrupt: 14
The error handler runs!
Unrestorable page fault! ( present read-only ) at 0x0xff10ffff
PANIC(Page fault) at paging.c:252
```

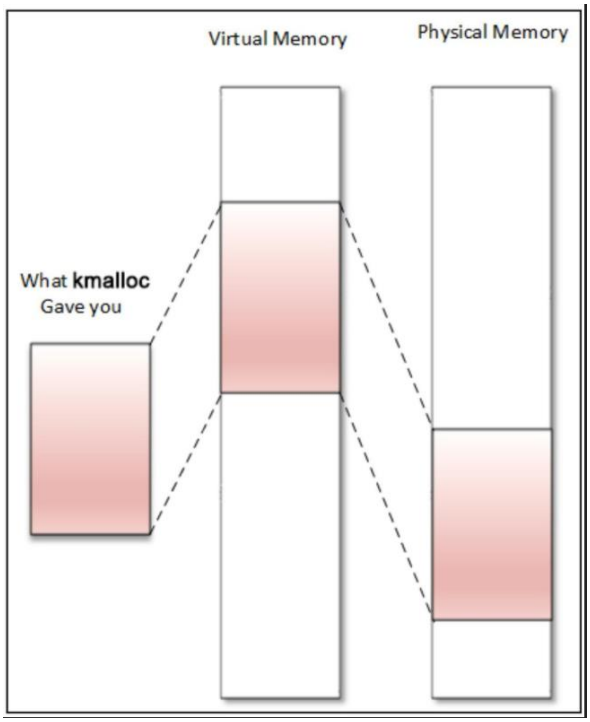
6. Сравнение MiniOS и UNIX:

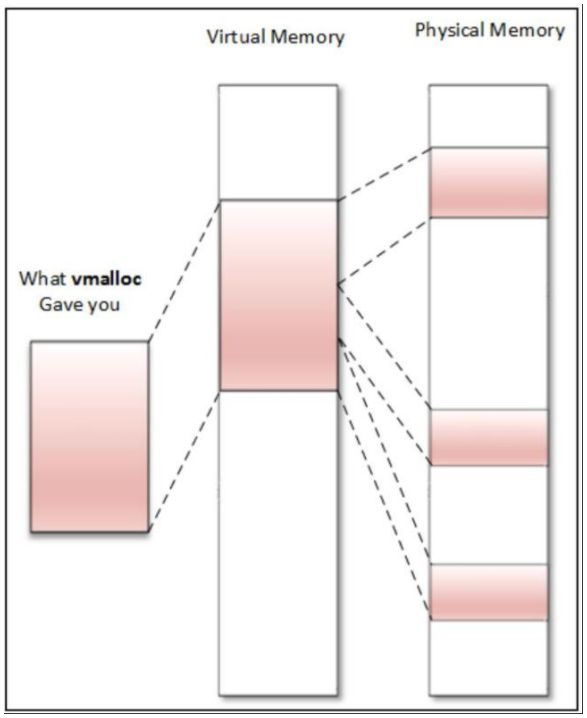
1) 1. kmalloc и vmalloc выделяют память ядра, malloc выделяет пользовательскую память.

2. kmalloc гарантирует, что выделенная память физически непрерывна. Malloc и vmalloc гарантируют непрерывность в виртуальном адресном пространстве.

3. Размер, который может выделить kmalloc, ограничен, а размер, который могут выделить vmalloc и malloc, относительно велик.

4. vmalloc медленнее, чем kmalloc. Хотя в некоторых случаях требуются только физически непрерывные блоки памяти, многие коды ядра используют kmalloc для получения памяти, а не vmalloc. В основном это связано с соображениями производительности. Чтобы преобразовать прерывистые страницы в физической памяти в непрерывные страницы в виртуальном адресном пространстве, функция vmalloc должна специально создавать записи таблицы страниц.





2) Структура виртуальной страницы в Linux в архитектуре Intel x86 совпадает с нашей структурой.

P - Установлено, если страница находится в памяти.
R/W - Если установлено, то на страницу можно выполнять запись. Если не установлено, то страница доступна только для чтения. Это поле не используется, когда код работает в режиме ядра (если не установлен флаг CR0).
U/S - Если установлено, то это пользовательский режим. В противном случае, это страница в режиме супервизора (ядра). Код пользовательского режима не может выполнять запись на страницы, находящиеся в режиме ядра, или читать из них.
Reserved - Для внутреннего использования процессором и это поле не следует менять.
A - Установлено, если к странице был доступ (выполнялось обращение процессора).
D - Установлено, если на станице выполнялась запись (страница изменена).
AVAIL - Эти три бита не используются и доступны в режиме ядра.
Page frame address - старшие 20 битов адреса фрейма в физической памяти.

```
typedef struct page
{
    u32int present    : 1;   //Присутствие страницы в памяти.
    u32int rw         : 1;   //Если сброшен, то страница только для чтения, если установлен, то страница для чтения и записи.
    u32int user       : 1;   //Если сброшен, то уровень супервизора.
    u32int accessed   : 1;   //Отображает, был ли доступ к странице с момента последнего обновления.
    u32int dirty      : 1;   //Отображает, производилась ли на страницу запись с момента последнего обновления.
    u32int unused     : 7;   //Объединение неиспользуемых и зарезервированных битов.
    u32int frame      : 20;  //Адрес фрейма(сдвинут вправо на 12 бит).
} page_t;
```

3) В нашем примере размер страниц составляет 4Кб, но в реальных системах используются размеры страниц от 512 байт до 1Гбайт. Например, архитектура x86-64 поддерживает размеры страницы размером 4Кбайт, 2 Мбайт и 1Гбайт.

Отличие логических адресов от виртуальных

Логические адреса

Они составляют обычное адресное пространство ядра. Эти адреса отображают какую-то часть (возможно, всю) основной памяти и часто рассматриваются, как если бы они были физическими адресами. На большинстве архитектур логические адреса и связанные с ними физические адреса отличаются только на постоянное смещение. Логические адреса используют родной размер указателя оборудования и, следовательно, могут быть не в состоянии адресовать всю физическую память на 32-х разрядных системах, оборудованных в большей степени. Память, возвращаемая kmalloc, имеет логический адрес ядра.

Виртуальные адреса

Виртуальные адреса ядра похожи на логические адреса в том, что они являются отображением адреса пространства ядра на физический адрес. Однако, виртуальные адреса ядра не всегда имеют линейную, взаимно-однозначную связь с физическими адресами, которая характеризует логическое адресное пространство. Все логические адреса являются виртуальными адресами ядра, но многие виртуальные адреса ядра не являются логическими адресами. Так, например, память, выделенная vmalloc, имеет виртуальный адрес (но без прямого физического отображения). Виртуальные адреса обычно хранятся в переменных указателей.

7.Выводы

В ходе выполнения задания по вычислительной практике мы изучили механизм прерываний в операционных системах и работу виртуальной памяти. Также мы познакомились с такими понятиями как виртуальное адресное пространство, page_fault.

Большинство систем виртуальной памяти использует технологию страничной организации памяти, где виртуальное адресное пространство поделено на блоки фиксированного размера, называемые страницами.

Для сопоставления виртуальным адресам физических адресов мы использовали программно-эмулируемый диспетчер памяти (MMU - Memory Management Unit).

В процессе работы мы научились создавать загрузочный образ операционной системы, благодаря чему мы смогли изучить работу MiniOs.

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 6.Paging [Электронный ресурс]: http://www.jamesmolloy.co.uk URL: http://www.jamesmolloy.co.uk/tutorial_html/5.-IRQs%20and%20the%20PIT.html
8. 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
9. Chapter 9.Multitasking [Электронный ресурс]: http://www.jamesmolloy.co.uk URL:http://www.jamesmolloy.co.uk/tutorial_html/9.-Multitasking.html
10. James Molloy's Tutorial Known Bugs [Электронный ресурс]: https://wiki.osdev.org URL:https://wiki.osdev.org/James_Molloy%27s_Tutorial_Known_Bugs
11. Язык Ада в проектировании систем/Р.Бар — Москва: Мир, 1988
12. Unix изнутри/Ю.Вахалия — Санкт-Петербург: Питер, 2003
13. X86 Assembly Language and C Fundamentals/J.Cavanagh — Лондон: CRC Press, 2013
14. OSDev.org [Электронный ресурс] URL:https://forum.osdev.org/
15. github.com [Электронный ресурс] URL:https://github.com/sukwon0709/osdev
16. Операционные системы/Э.Танненбаум, А.Вудхалл — Санкт-Петербург: Питер, 2007
17. Пособие по разработке макросов NASM
18. Interactive map of Linux kernel [Электронный ресурс]:www.makelinux.net URL:http://www.makelinux.net/kernel_map/
19. Wikipedia INT(x86 instruction) [Электронный ресурс] URL:https://en.wikipedia.org/wiki/INT_(x86_instruction)
20. Wikipedia Interrupt descriptor table [Электронный ресурс] URL:https://en.wikipedia.org/wiki/Interrupt_descriptor_table