|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ | | | | | | | | |
| **РОССИЙСКОЙ ФЕДЕРАЦИИ** | | | | | | | | |
|  |  | | |  |  | |  | |
|  |  | | |  |  | |  | |
|  | |  |  | | |  | |  |
| **МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)** | | | | | | | | |
|  | | | | | | | | |

## **ЖУРНАЛ**

#### ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Наименование практики *вычислительная*

Студенты :

*Аксенов Александр Евгеньевич*

*Катермин Всеволод*

*Хисамутдинов Даниил*

Факультет № 8 курс 2 группа 8

Практика с 22.06.20 по 12.07.20

##### 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 2020г.

Студент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (подпись) Студент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (подпись) Студент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (подпись)

**ЗАДАНИЕ**

кафедры 806 по вычислительной практике

1. Познакомиться с загрузчиком GRUB. Создать загрузочную флешку с образом ядра MiniOS и проверить его целостность.
2. Изучить архитектуру MiniOS и разобраться в механизме прерываний.
3. Составить алгоритм **управления защищенным и пользовательским режимом ядра**.
4. Реализация алгоритма.
5. Тестирование алгоритма.
6. Список используемой литературы.
7. Выводы.

**Руководитель практики**

**от института**

«12» июля 2020 г.

**ПРОТОКОЛ**

#### ЗАЩИТЫ ТЕХНИЧЕСКОГО ОТЧЁТА

по *вычислительной практике*

студентами 1. *Аксеновым Александром*

*2. Катерминым Всеволодом*

*3. Хисамутдиновым Даниилом*

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

Руководитель: Семенов А. С. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 12.07.2020

1. Инициализация таблиц gdt idt, файловой системы и переключение в пользовательских режим
2. Инициализация процессов, paging, tasking
3. Вызов прерывания для перехода к планировщику
4. Обращение к планировщику
5. Генерация прерывания для первого процесса
6. Возврат в первый процесс
7. Вызов прерывания для смены процессов
8. Обращение к планировщику
9. Генерация прерывания для второго процесса
10. Передача управления второму процессу
11. Вызов прерывания
12. Возврат в main
13. Завершение функции main

Планировщик

**Алгоритм. Взаимодействие пользовательских процессов**

process1

Клавиатура

Таймер

Контроллер прерываний

**Процессор**

Process2

main

Касаемо режимов пользователя и ядра:

Для предотвращения доступа приложений к критически важным данным операционной системы и устранения риска их модификации используются два режима доступа к процессору: пользовательский (user mode) и ядра (kernel mode). Код приложений работает в пользовательском режиме, тогда как код операционной системы (например, системные сервисы и драйверы устройств) — в режиме ядра. B режиме ядра предоставляется доступ ко всей системной памяти и разрешается выполнять любые машинные команды процессора. Предоставляя операционной системе более высокий уровень привилегий, чем прикладным программам, процессор позволяет разработчикам операционных систем реализовать такие архитектуры, которые не дают возможности сбойным приложениям нарушать стабильность работы всей системы.

Существует четыре уровня привилегий (PL) сегментов (0-3). Привилегированность увеличивается с уменьшением номера. На нулевом уровне позволяется использование привилегированных инструкций.

Уровень привилегий сегмента (Descriptor Privilege Level) соответствует значению поля DPL в дескрипторе сегмента.

Текущий уровень привилегий (Current Privilege Level) соответствует уровню привилегий сегмента кода, селектор которого загружен в регистр CS (то есть уровню привилегий выполняющегося сегмента кода).

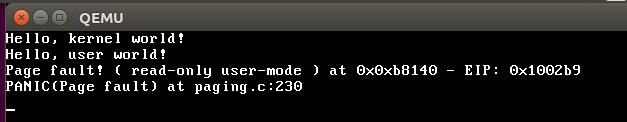
Запрашиваемый уровень привилегий (Requested Privilege Level) находится в двух младших битах селектора (задаётся программой).

**ТЕХНИЧЕСКИЙ ОТЧЁТ ПО ПРАКТИКЕ**

**4. Реализация процедуры:**

|  |  |  |
| --- | --- | --- |
| № | Шаг алгоритма | Файл и код |
| 1 | Инициализация таблиц дескрипторов, прерываний, страничной памяти, мультизадачности, файловой системы, процессов. | ***main.c***  …  init\_descriptor\_tables();  monitor\_clear();  initialise\_paging();  initialise\_tasking();  fs\_root = initialise\_initrd(initrd\_location);  initialise\_syscalls();  switch\_to\_user\_mode();  … |
| 2 | Загрузка адреса таблиц прерываний, дескрипторов и состояний задач в процессор | ***gdt.s***  …  [GLOBAL gdt\_flush] ; Allows the C code to call gdt\_flush().  …  [GLOBAL idt\_flush] ; Allows the C code to call idt\_flush().  …  [GLOBAL tss\_flush] ; Allows our C code to call tss\_flush().  … |
| 3 | Вызов первого прерывания | ***main.c***  …  asm volatile("int $0x0");  … |
| 4 | Обработка прерывания, сохранение регистров предыдущей задачи, загрузка адреса инструкции обработки прерывания в регистры, загрузка регистров. | ***interrupt.s***  …  isr\_common\_stub:  pusha ; Pushes edi,esi,ebp,esp,ebx,edx,ecx,eax  mov ax, ds ; Lower 16-bits of eax = ds.  push eax ; save the data segment descriptor  mov ax, 0x10 ; load the kernel data segment descriptor  mov ds, ax  mov es, ax  mov fs, ax  mov gs, ax  call isr\_handler ;interrupt handler  …  ***isr.c***  …  void isr\_handler(registers\_t regs)  {  monitor\_write("recieved interrupt: ");  monitor\_write\_dec(regs.int\_no);  monitor\_put('\n');  if (interrupt\_handlers[regs.int\_no] != 0)  {  isr\_t handler = interrupt\_handlers[regs.int\_no];  //вызывается функция обработки прерывания  handler(regs);  }  }  … |
| 5 | Посредством функции обработки прерывания происходит взаимодействие с файловой системой | ***main.c***  …  initialise\_syscalls();  syscall\_monitor\_write("Hello, core world!\n");  switch\_to\_user\_mode();  … |
| 6 | Завершение обработки прерывания, восстановление регистров, возврат к предыдущей задаче | ***interrupt.s***  …  pop ebx ; reload the original data segment descriptor  mov ds, bx  mov es, bx  mov fs, bx  mov gs, bx  popa ; Pops edi,esi,ebp...  add esp, 8 ; Cleans up the pushed error code and pushed  ; ISR number  sti  iret ; pops 5 things at once: CS, EIP, EFLAGS, SS, and  ; ESP  … |

**5. Тестирование программы.**



**Файлы**

**task.c**

void switch\_to\_user\_mode()

{

// Настраиваем структуру стека для переключения в пользовательский режим.

asm volatile(" \

cli; \

mov $0x23, %ax; \

mov %ax, %ds; \

mov %ax, %es; \

mov %ax, %fs; \

mov %ax, %gs; \

\

mov %esp, %eax; \

pushl $0x23; \

pushl %eax; \

pushf; \

pop %eax ; Помещаем EFLAGS обратно в EAX. Единственный способ — читаем EFLAGS с помощью pushf, а //затем выталкиваем из стека.

or %eax, $0x200 ; Устанавливаем флаг IF.

push %eax ; Помещаем новое значение EFLAGS обратно в стек.

pushl $0x1B; \

push $1f; \

iret; \

1: \

");

}

**descriptor\_tables.h**

// Структура, описывающая сегмент состояния задачи Task State Segment.

struct tss\_entry\_struct

{

u32int prev\_tss; // Предыдущий TSS – если используется аппаратное переключение задач, то это поле нужно создания связного списка.

u32int esp0; // Указатель стека, загружаемый при переходе в режим ядра.

u32int ss0; // Сегмент стека, загружаемый при переходе в режим ядра.

u32int esp1; // Не используется ...

u32int ss1;

u32int esp2;

u32int ss2;

u32int cr3;

u32int eip;

u32int eflags;

u32int eax;

u32int ecx;

u32int edx;

u32int ebx;

u32int esp;

u32int ebp;

u32int esi;

u32int edi;

u32int es; // Значение, загружаемое в ES при переходе в режим ядра.

u32int cs; // Значение, загружаемое в CS при переходе в режим ядра

u32int ss; // Значение, загружаемое в SS при переходе в режим ядра

u32int ds; // Значение, загружаемое в DS при переходе в режим ядра

u32int fs; // Значение, загружаемое в FS при переходе в режим ядра

u32int gs; // Значение, загружаемое в GS при переходе в режим ядра

u32int ldt; // Не используется ...

u16int trap;

u16int iomap\_base;

} \_\_attribute\_\_((packed));

typedef struct tss\_entry\_struct tss\_entry\_t;

**descriptor\_tables.c**

// Давайте из кода на языке C получим доступ к нашим ассемблерным функциям.

...

extern void tss\_flush();

// Внутренние прототипы функций.

...

static void write\_tss(s32int,u16int,u32int);

...

tss\_entry\_t tss\_entry;

static void init\_gdt()

{

gdt\_ptr.limit = (sizeof(gdt\_entry\_t) \* 6) - 1;

gdt\_ptr.base = (u32int)&gdt\_entries;

gdt\_set\_gate(0, 0, 0, 0, 0); // Сегмент null

gdt\_set\_gate(1, 0, 0xFFFFFFFF, 0x9A, 0xCF); // Сегмент кода

gdt\_set\_gate(2, 0, 0xFFFFFFFF, 0x92, 0xCF); // Сегмент данных

gdt\_set\_gate(3, 0, 0xFFFFFFFF, 0xFA, 0xCF); // Сегмент кода пользовательского режима

gdt\_set\_gate(4, 0, 0xFFFFFFFF, 0xF2, 0xCF); // Сегмент данных пользовательского режима

write\_tss(5, 0x10, 0x0);

gdt\_flush((u32int)&gdt\_ptr);

tss\_flush();

}

// Инициализируем нашу структуру сегмента состояния задачи.

static void write\_tss(s32int num, u16int ss0, u32int esp0)

{

// Сначала давайте вычислим базу и предельное значение для нашей записи в таблице GDT.

u32int base = (u32int) &tss\_entry;

u32int limit = base + sizeof(tss\_entry);

// Теперь добавим в таблицу GDT адрес дескриптора нашего TSS.

gdt\_set\_gate(num, base, limit, 0xE9, 0x00);

// Обеспечим, чтобы первоначально дескриптор был равен нулю.

memset(&tss\_entry, 0, sizeof(tss\_entry));

tss\_entry.ss0 = ss0; // Запоминаем сегмент стека ядра.

tss\_entry.esp0 = esp0; // Запоминаем указатель стека ядра.

// Здесь заносим в таблицу TSS записи cs, ss, ds, es, fs и gs. В них указывается, какие сегменты

// должны быть загружены в случае, когда процессор переключается в режим ядра. Поэтому

// они являются нашими обычными сегментами кода/данных ядра - 0x08 и 0x10 соответственно,

// но в последних двух битах будут указаны значения 0x0b и 0x13. Значения этих битов указывают,

// что уровень запрашиваемых привилегий RPL (requested privilege level) равен 3; это означает, что

// этот сегмент TSS можно использовать для переключения в режим ядра из кольца 3.

tss\_entry.cs = 0x0b;

tss\_entry.ss = tss\_entry.ds = tss\_entry.es = tss\_entry.fs = tss\_entry.gs = 0x13;

}

void set\_kernel\_stack(u32int stack)

{

tss\_entry.esp0 = stack;

}

**gdt.c**

[GLOBAL tss\_flush] ; Allows our C code to call tss\_flush().

tss\_flush:

mov ax, 0x2B ; Load the index of our TSS structure - The index is

; 0x28, as it is the 5th selector and each is 8 bytes

; long, but we set the bottom two bits (making 0x2B)

; so that it has an RPL of 3, not zero.

ltr ax ; Load 0x2B into the task state register.

ret

**syscall.h**

// syscall.h – Определяет интерфейс и структуры, относящиеся к диспетчеризации системных вызовов.

// Написано для руководств по разработке ядра - автор James Molloy

#ifndef SYSCALL\_H

#define SYSCALL\_H

#define DECL\_SYSCALL0(fn) int syscall\_##fn();

#define DECL\_SYSCALL1(fn,p1) int syscall\_##fn(p1);

#define DECL\_SYSCALL2(fn,p1,p2) int syscall\_##fn(p1,p2);

#define DECL\_SYSCALL3(fn,p1,p2,p3) int syscall\_##fn(p1,p2,p3);

#define DECL\_SYSCALL4(fn,p1,p2,p3,p4) int syscall\_##fn(p1,p2,p3,p4);

#define DECL\_SYSCALL5(fn,p1,p2,p3,p4,p5) int syscall\_##fn(p1,p2,p3,p4,p5);

#define DEFN\_SYSCALL0(fn, num) \

int syscall\_##fn() \

{ \

int a; \

asm volatile("int $0x80" : "=a" (a) : "0" (num)); \

return a; \

}

#define DEFN\_SYSCALL1(fn, num, P1) \

int syscall\_##fn(P1 p1) \

{ \

int a; \

asm volatile("int $0x80" : "=a" (a) : "0" (num), "b" ((int)p1)); \

return a; \

}

#define DEFN\_SYSCALL2(fn, num, P1, P2) \

int syscall\_##fn(P1 p1, P2 p2) \

{ \

int a; \

asm volatile("int $0x80" : "=a" (a) : "0" (num), "b" ((int)p1), "c" ((int)p2)); \

return a; \

}

DECL\_SYSCALL1(monitor\_write, const char\*)

DECL\_SYSCALL1(monitor\_write\_hex, const char\*)

DECL\_SYSCALL1(monitor\_write\_dec, const char\*)

#include "common.h"

void initialise\_syscalls();

#endif

**syscall.c**

// syscall.c – Определяет реализацию механизма системных вызовов.

// Написано для руководств по разработке ядра - автор James Molloy

#include "syscall.h"

#include "isr.h"

#include "monitor.h"

DEFN\_SYSCALL1(monitor\_write, 0, const char\*);

DEFN\_SYSCALL1(monitor\_write\_hex, 1, const char\*);

DEFN\_SYSCALL1(monitor\_write\_dec, 2, const char\*);

static void syscall\_handler(registers\_t \*regs);

static void \*syscalls[3] =

{

&monitor\_write,

&monitor\_write\_hex,

&monitor\_write\_dec,

};

u32int num\_syscalls = 3;

void initialise\_syscalls()

{

// Регистрируем наш обработчик системных вызовов.

register\_interrupt\_handler (0x80, &syscall\_handler);

}

void syscall\_handler(registers\_t \*regs)

{

// Сначала проверяем, является ли допустимым запрашиваемый номер системного вызова.

// Номер системного вызова находится в EAX.

if (regs->eax >= num\_syscalls)

return;

// Вычисляем место, где находится запрашиваемый системный вызов.

void \*location = syscalls[regs->eax];

// Нам неизвестно, сколько параметров необходимо функции, поэтому мы просто

// помещаем их в стек в правильном порядке. Функция может использовать все эти

// параметры, если они потребуются, а затем мы можем убрать их из стека.

int ret;

asm volatile (" \

push %1; \

push %2; \

push %3; \

push %4; \

push %5; \

call \*%6; \

pop %%ebx; \

pop %%ebx; \

pop %%ebx; \

pop %%ebx; \

pop %%ebx; \

" : "=a" (ret) : "r" (regs->edi), "r" (regs->esi), "r" (regs->edx), "r" (regs->ecx), "r" (regs->ebx), "r" (location));

regs->eax = ret;

}

**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](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](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](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](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](https://wiki.osdev.org/) URL:[https://wiki.osdev.org/James\_Molloy%27s\_Tutorial\_Known\_Bugs](https://wiki.osdev.org/James_Molloy's_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](http://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>)

1. 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. Так как для создания сборки MiniOS с виртуальной файловой системой нужно подгрузить модуль initrd, то в папке grub в menu.lst под строкой, начинающейся с "kernel", дописываем "module /initrd"
4. Долго не получалось подключить виртуальную файловую систему из-за неправильной инструкции сборки.
5. В процессе выполнения задания изучили механизм прерываний.
6. Каждую из сборок легко собрать, немного изменив первоначальный make файл.
7. Много полезной информации нашлось на иностранных ресурсах.