МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЖУРНАЛ
ПО ВЫЧИСЛИТЕЛЬНОЙ ПРАКТИКЕ

Наименование практики *вычислительная*

Студенты:

Демина Виктория Алексеевна
Айрапетова Евгения Ашотовна
Пивницкий Даниэль Сергеевич

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 2021 г.

Студент Демина Виктория Алексеевна (подпись)

Студент Айрапетова Евгения Ашотовна (подпись)

Студент Пивницкий Даниэль Сергеевич (подпись)

ЗАДАНИЕ

кафедры 806 по вычислительной практике

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.

Руководитель практики
от института

«12» июля 2020 г. **Подпись**

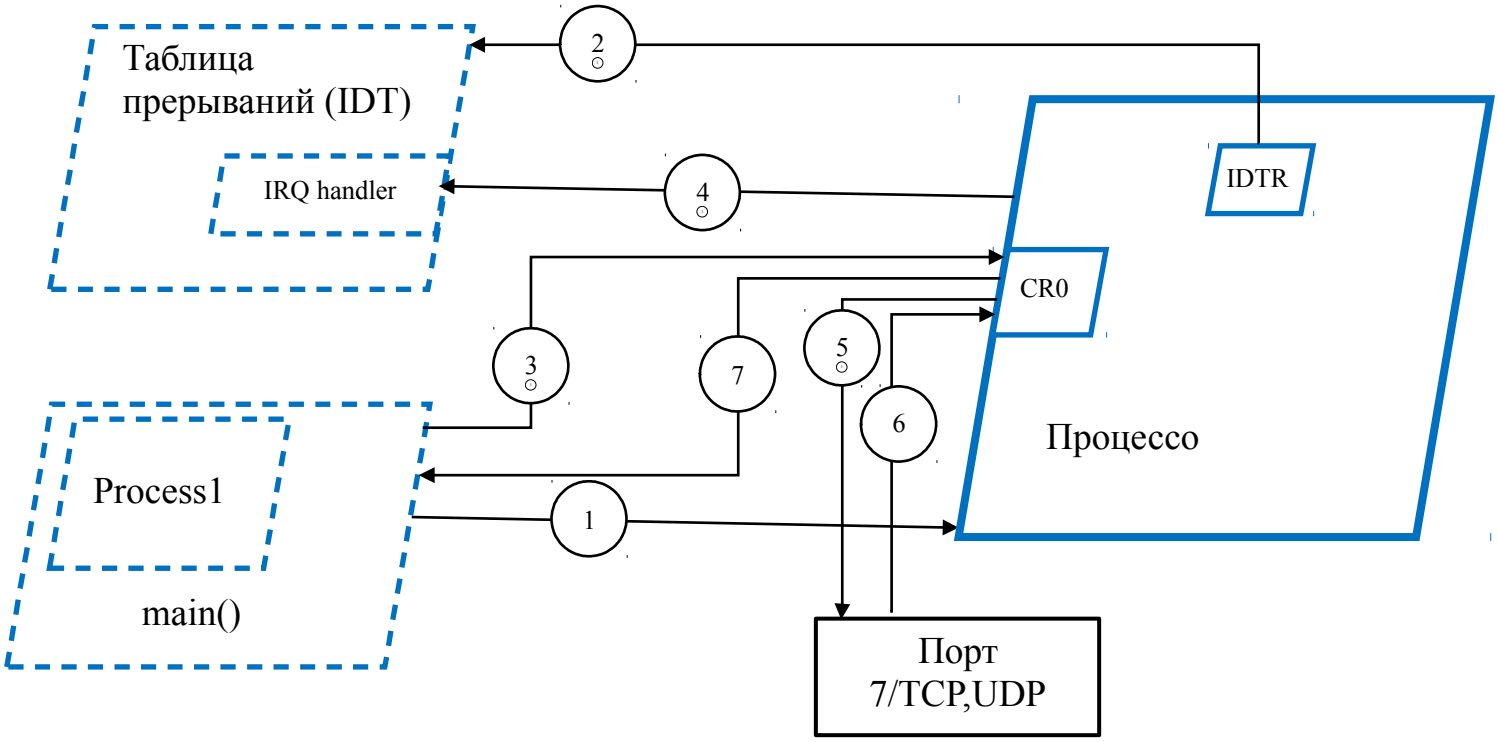
ПРОТОКОЛ
ЗАЩИТЫ ТЕХНИЧЕСКОГО ОТЧЁТА

по *вычислительной практике*
студентами

1. *Демина Виктория Алексеевна*
2. *Айрапетовой Евгения Ашотовна*
3. *Пивницкий Даниэль Сергеевич*

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 алгоритм взаимодействия пользовательских процессов.	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: 12.07.2020



Алгоритм. Управление сетевым интерфейсом.

1. Инициализация таблицы прерываний, Process1.
2. Инициализация обработчиков прерываний, сохранение адреса таблицы прерываний в IDTR (?)
3. Из main процесса поступает сигнал, вызывающий прерывание
4. Обработка прерывания. Запись информации о прерывании в IDT
5. Передача данных в порт
6. Получение данных из порта в неизменном виде
7. Завершение обработки прерываний, возврат к выполнению основного процесса

Сетевой интерфейс в Linux.

Сетевой интерфейс - это физическое или виртуальное устройство, предназначенное для передачи данных между программами через компьютерную сеть. В данном случае данные передаются между компьютером пользователя и сетью.

Сетевое взаимодействие Linux-компьютера происходит через сетевые интерфейсы. Любые данные, которые компьютер отправляет в сеть или получает из сети проходят через сетевой интерфейс.

Интерфейс определён реализацией модели TCP/IP для того чтобы скрыть различия в сетевом обеспечении и свести сетевое взаимодействие к обмену данными с абстрактной сущностью.

Технический отчет по практике

Реализация процедуры:

№	Шаг алгоритма	Файл и код

1.	Инициализация таблиц дескрипторов прерываний, процессов	**main.c**
		`static int igb_request_irq(struct igb_adapter *adapter)`
		`{`
		`struct net_device *netdev = adapter->netdev;`
		`struct pci_dev *pdev = adapter->pdev;`
		`int err = 0;`
		
		`if (adapter->msix_entries) {`
		`err = igb_request_msix(adapter);`
		`if (!err)`
		`goto request_done;`

2.	Инициализация обработчиков прерываний, сохранение адреса таблицы в IDTR	**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;

3.	Из main процесса поступает сигнал, вызывающий прерывание	**main.c**
		static int igb_request_msix(struct igb_adapter *adapter)
		{
		struct net_device *netdev = adapter->netdev;
		struct e1000_hw *hw = &adapter->hw;
		int i, err = 0, vector = 0, free_vector = 0;
		
		err = request_irq(adapter->msix_entries[vector].vector,
		igb_msix_other, 0, netdev->name, adapter);
		if (err)
		goto err_out;

4.	Обработка прерывания. Запись информации о прерывании в IDT	**main.c**
		static irqreturn_t igb_intr_msi(int irq, void *data)
		{
		struct igb_adapter *adapter = data;
		struct igb_q_vector *q_vector = adapter->q_vector[0];
		struct e1000_hw *hw = &adapter->hw;

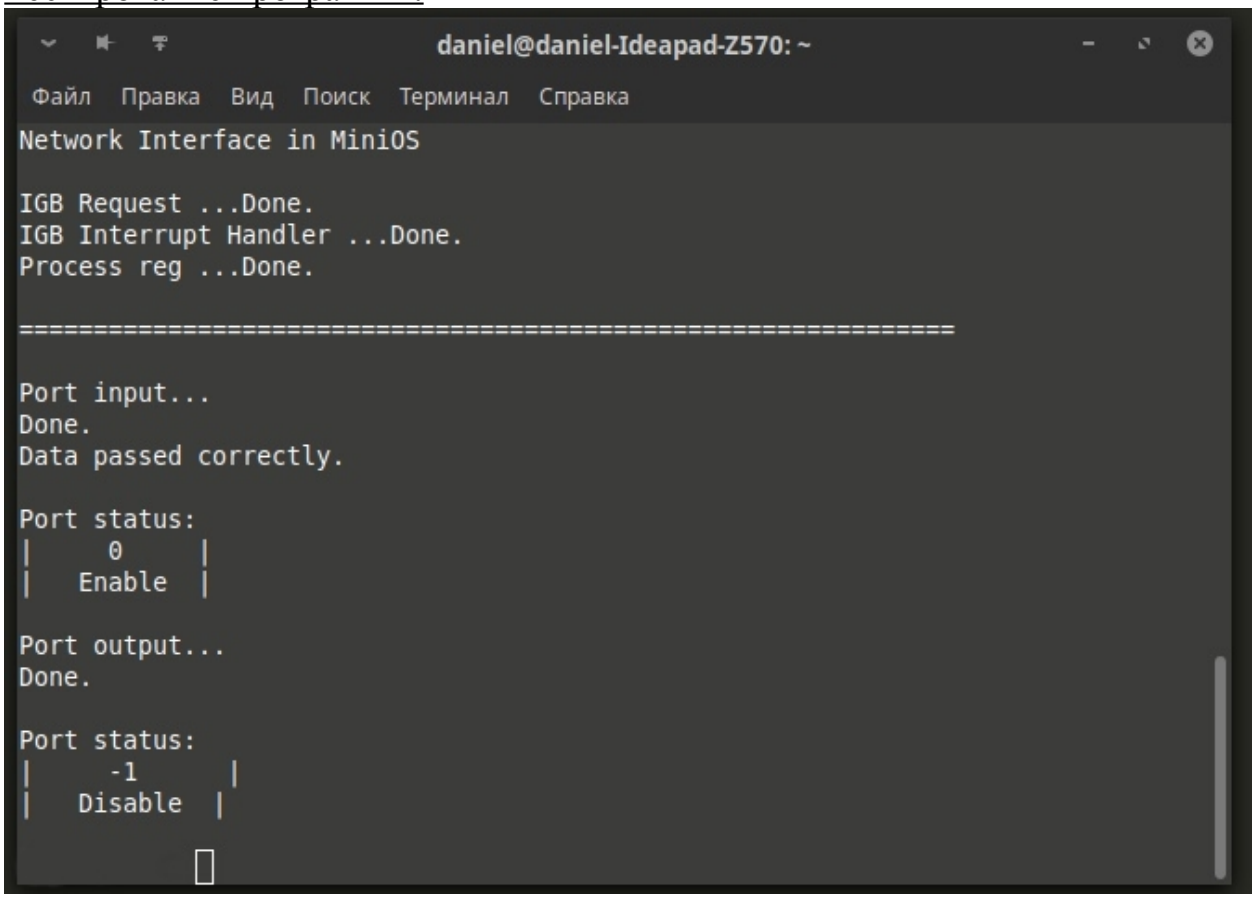
5.	Передача данных в порт	**port.h** enum sci_status sci_port_add_phy(struct isci_port *iport, struct isci_phy *iphy); void sci_port_setup_transports(struct isci_port *iport, u32 device_id);

6.	Получение данных из порта в неизменном виде	**port.h**
		struct sci_port_end_point_properties {
		struct sci_sas_address sas_address;
		struct sci_phy_proto protocols;
		};
		
		struct sci_port_properties {
		u32 index;
		struct sci_port_end_point_properties local;
		struct sci_port_end_point_properties remote;
		u32 phy_mask;
		};

7.	Завершение обработки	**interrupt.s** … pop ebx ; reload the original data segment

	прерываний, возврат к выполнению основного процесса	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
		…

Тестирование программы:



Файлы программы:

main.c

```
#include <linux/i2c.h>
#include "igb.h"
#include "port.h"
#include "monitor.h"
#include "multiboot.h"
#include "descriptor_tables.h"
#include "timer.h"

/**
 *  igb_request_irq - initialize interrupts
 *  @adapter: board private structure to initialize
 *
 *  Attempts to configure interrupts using the best available
 *  capabilities of the hardware and kernel.
 **/
static int igb_request_irq(struct igb_adapter *adapter)
{
      struct net_device *netdev = adapter->netdev;
      struct pci_dev *pdev = adapter->pdev;
      int err = 0;

      if (adapter->msix_entries) {
            err = igb_request_msix(adapter);
            if (!err)
                  goto request_done;
            /* fall back to MSI */
            igb_free_all_tx_resources(adapter);
            igb_free_all_rx_resources(adapter);

            igb_clear_interrupt_scheme(adapter);
            err = igb_init_interrupt_scheme(adapter, false);
            if (err)
                  goto request_done;

            igb_setup_all_tx_resources(adapter);
            igb_setup_all_rx_resources(adapter);
            igb_configure(adapter);
      }
```

```
	igb_assign_vector(adapter->q_vector[0], 0);

	if (adapter->flags & IGB_FLAG_HAS_MSI) {
		err = request_irq(pdev->irq, igb_intr_msi, 0,
				  netdev->name, adapter);
		if (!err)
			goto request_done;

		/* fall back to legacy interrupts */
		igb_reset_interrupt_capability(adapter);
		adapter->flags &= ~IGB_FLAG_HAS_MSI;
	}

	err = request_irq(pdev->irq, igb_intr, IRQF_SHARED,
			  netdev->name, adapter);

	if (err)
		dev_err(&pdev->dev, "Error %d getting interrupt\n",
			err);

request_done:
	return err;
}

/**
 *  igb_request_msix - Initialize MSI-X interrupts
 *  @adapter: board private structure to initialize
 *
 *  igb_request_msix allocates MSI-X vectors and requests interrupts from the
 *  kernel.
 **/
static int igb_request_msix(struct igb_adapter *adapter)
{
	struct net_device *netdev = adapter->netdev;
	struct e1000_hw *hw = &adapter->hw;
	int i, err = 0, vector = 0, free_vector = 0;
```

```
	err = request_irq(adapter->msix_entries[vector].vector,
	                  igb_msix_other, 0, netdev->name, adapter);
	if (err)
		goto err_out;

	for (i = 0; i < adapter->num_q_vectors; i++) {
		struct igb_q_vector *q_vector = adapter->q_vector[i];

		vector++;

		q_vector->itr_register = hw->hw_addr + E1000_EITR(vector);

		if (q_vector->rx.ring && q_vector->tx.ring)
			sprintf(q_vector->name, "%s-TxRx-%u", netdev->name,
				q_vector->rx.ring->queue_index);
		else if (q_vector->tx.ring)
			sprintf(q_vector->name, "%s-tx-%u", netdev->name,
				q_vector->tx.ring->queue_index);
		else if (q_vector->rx.ring)
			sprintf(q_vector->name, "%s-rx-%u", netdev->name,
				q_vector->rx.ring->queue_index);
		else
			sprintf(q_vector->name, "%s-unused", netdev->name);

		err = request_irq(adapter->msix_entries[vector].vector,
				  igb_msix_ring, 0, q_vector->name,
				  q_vector);
		if (err)
			goto err_free;
	}

	igb_configure_msix(adapter);
	return 0;

err_free:
	/* free already assigned IRQs */
	free_irq(adapter->msix_entries[free_vector++].vector, adapter);
```

```
	vector--;
	for (i = 0; i < vector; i++) {
		free_irq(adapter->msix_entries[free_vector++].vector,
			 adapter->q_vector[i]);
	}
err_out:
	return err;
}

/**
 *  igb_irq_disable - Mask off interrupt generation on the NIC
 *  @adapter: board private structure
 **/
static void igb_irq_disable(struct igb_adapter *adapter)
{
	struct e1000_hw *hw = &adapter->hw;

	/* we need to be careful when disabling interrupts.  The VFs are also
	 * mapped into these registers and so clearing the bits can cause
	 * issues on the VF drivers so we only need to clear what we set
	 */
	if (adapter->msix_entries) {
		u32 regval = rd32(E1000_EIAM);
		wr32(E1000_EIAM, regval & ~adapter->eims_enable_mask);
		wr32(E1000_EIMC, adapter->eims_enable_mask);
		regval = rd32(E1000_EIAC);
		wr32(E1000_EIAC, regval & ~adapter->eims_enable_mask);
	}

	wr32(E1000_IAM, 0);
	wr32(E1000_IMC, ~0);
	wrfl();
	if (adapter->msix_entries) {
		int i;
		for (i = 0; i < adapter->num_q_vectors; i++)
			synchronize_irq(adapter->msix_entries[i].vector);
	} else {
		synchronize_irq(adapter->pdev->irq);
```

```
	}
}

/**
 *  igb_intr_msi - Interrupt Handler
 *  @irq: interrupt number
 *  @data: pointer to a network interface device structure
 **/
static irqreturn_t igb_intr_msi(int irq, void *data)
{
	struct igb_adapter *adapter = data;
	struct igb_q_vector *q_vector = adapter->q_vector[0];
	struct e1000_hw *hw = &adapter->hw;
	/* read ICR disables interrupts using IAM */
	u32 icr = rd32(E1000_ICR);

	igb_write_itr(q_vector);

	if (icr & E1000_ICR_DRSTA)
		schedule_work(&adapter->reset_task);

	if (icr & E1000_ICR_DOUTSYNC) {
		/* HW is reporting DMA is out of sync */
		adapter->stats.doosync++;
	}

	if (icr & (E1000_ICR_RXSEQ | E1000_ICR_LSC)) {
		hw->mac.get_link_status = 1;
		if (!test_bit(__IGB_DOWN, &adapter->state))
			mod_timer(&adapter->watchdog_timer, jiffies + 1);
	}

	if (icr & E1000_ICR_TS) {
		u32 tsicr = rd32(E1000_TSICR);

		if (tsicr & E1000_TSICR_TXTS) {
			/* acknowledge the interrupt */
			wr32(E1000_TSICR, E1000_TSICR_TXTS);
```

```
                /* retrieve hardware timestamp */
                schedule_work(&adapter->ptp_tx_work);
            }
        }

        napi_schedule(&q_vector->napi);

        return IRQ_HANDLED;
}

void init_process_one(){
   register_interrupt_handler(0x0, &process_one); //регистрируем первый
процесс с номером прерывания 0
}
static void process_two(registers_t reg){

   int flags = 1;
   int fd;
   char buf[256];
   char* str = "New text for test2.txt";

monitor_write("Opening test2.txt\n");
   if((fd = open("test2.txt", flags))>0){
monitor_write("File test2.txt opened!\n");
}
monitor_write("Reading the second file...\n");
read(fd, buf, 256,0);
monitor_write("\t");
monitor_write(buf);
monitor_write("\n");
monitor_write("Writing to second file...\n");
write(fd,str,256,0);
monitor_write("Reading the second file...\n");
read(fd, buf, 256,0);
monitor_write("\t");
monitor_write(buf);
monitor_write("\n");
monitor_write("Closing the second file...\n");
```

```
close(fd);
}

void init_process_two(){
   register_interrupt_handler(0x1, &process_two);//регистрируем второй процесс с номером прерывания
}
/**
 *  igb_intr_close - Disables a network interface
 *  @netdev: network interface device structure
 *
 *  Returns 0, this is not allowed to fail
 *
 *  The close entry point is called when an interface is de-activated
 *  by the OS.  The hardware is still under the driver's control, but
 *  needs to be disabled.  A global MAC reset is issued to stop the
 *  hardware, and all transmit and receive resources are freed.
 **/
static int __igb_intr_close(struct net_device *netdev, bool suspending)
{
      struct igb_adapter *adapter = netdev_priv(netdev);
      struct pci_dev *pdev = adapter->pdev;

      WARN_ON(test_bit(__IGB_RESETTING, &adapter->state));

      if (!suspending)
            pm_runtime_get_sync(&pdev->dev);

      igb_down(adapter);
      igb_free_irq(adapter);

      igb_free_all_tx_resources(adapter);
      igb_free_all_rx_resources(adapter);

      if (!suspending)
            pm_runtime_put_sync(&pdev->dev);
      return 0;
}
```

```
static int igb_intr_close(struct net_device *netdev)
{
      return __igb_intr_close(netdev, false);
}


int main (struct multiboot *mboot_ptr){
    int igb_request_irq(); //инициализируем прерывание
    int igb_request_msix(); // поступает сигнал, вызывающий прерывание
    void igb_irq_disable(); //отключение генерации прерывания
    irqreturn_t igb_intr_msi(); //обработка прерывания
    void init_process_one();////регистрируем прерывания в IDT
    void init_process_two();//
    struct sci_port_end_point_properties();//передача данных в порт
    struct sci_port_properties ());//получение данных из порта в неизменном
виде
    monitor_write("\nDone!");
    return 0;
}
```

port.h

```
#ifndef _ISCI_PORT_H_
#define _ISCI_PORT_H_

#include <iostream>
#include <scsi/libsas.h>
#include "isci.h"
#include "sas.h"
#include "phy.h"

#define SCIC_SDS_DUMMY_PORT 0xFF

#define PF_NOTIFY (1 << 0)
#define PF_RESUME (1 << 1)

struct isci_phy;
struct isci_host;

enum isci_status {
isci_freed = 0x00,
isci_starting = 0x01,
isci_ready = 0x02,
isci_ready_for_io = 0x03,
```

```
isci_stopping = 0x04,
isci_stopped = 0x05,
};
struct isci_port {
struct isci_host *isci_host;
struct list_head remote_dev_list;
#define IPORT_RESET_PENDING 0
unsigned long state;
enum sci_status hard_reset_status;
struct sci_base_state_machine sm;
bool ready_exit;
u8 logical_port_index;
u8 physical_port_index;
u8 active_phy_mask;
u8 enabled_phy_mask;
u8 last_active_phy;
u16 reserved_rni;
u16 reserved_tag;
u32 started_request_count;
u32 assigned_device_count;
u32 hang_detect_users;
u32 not_ready_reason;
struct isci_phy *phy_table[SCI_MAX_PHYS];
struct isci_host *owning_controller;
struct sci_timer timer;
struct scu_port_task_scheduler_registers __iomem
*port_task_scheduler_registers;
/* XXX rework: only one register, no need to replicate per-port */
u32 __iomem *port_pe_configuration_register;
struct scu_viit_entry __iomem *viit_registers;
};

enum sci_port_not_ready_reason_code {
SCIC_PORT_NOT_READY_NO_ACTIVE_PHYS,
SCIC_PORT_NOT_READY_HARD_RESET_REQUESTED,
SCIC_PORT_NOT_READY_INVALID_PORT_CONFIGURATION,
SCIC_PORT_NOT_READY_RECONFIGURING,

SCIC_PORT_NOT_READY_REASON_CODE_MAX
};

struct sci_port_end_point_properties {
struct sci_sas_address sas_address;
struct sci_phy_proto protocols;
};

struct sci_port_properties {
u32 index;
```

```
struct sci_port_end_point_properties local;
struct sci_port_end_point_properties remote;
u32 phy_mask;
};
#define PORT_STATES {\

C(PORT_STOPPED),\
C(PORT_STOPPING),\
C(PORT_READY),\
C(PORT_SUB_WAITING),\
C(PORT_SUB_OPERATIONAL),\
C(PORT_SUB_CONFIGURING),\
C(PORT_RESETTING),\
C(PORT_FAILED),\
}
#undef C
#define C(a) SCI_##a
enum sci_port_states PORT_STATES;
#undef C

static inline void sci_port_decrement_request_count(struct isci_port *iport)
{
if (WARN_ONCE(iport->started_request_count == 0,
"%s: tried to decrement started_request_count past 0!?",
__func__))
/* pass */;
else
iport->started_request_count--;
}

#define sci_port_active_phy(port, phy) \
(((port)->active_phy_mask & (1 << (phy)->phy_index)) != 0)

void sci_port_construct(
struct isci_port *iport,
u8 port_index,
struct isci_host *ihost);

enum sci_status sci_port_start(struct isci_port *iport);
enum sci_status sci_port_stop(struct isci_port *iport);

enum sci_status sci_port_add_phy(
struct isci_port *iport,
struct isci_phy *iphy);

void sci_port_setup_transports(
struct isci_port *iport,
u32 device_id);
```

```
enum sci_status sci_port_remove_phy(
struct isci_port *iport,
struct isci_phy *iphy);

void isci_port_bcn_enable(struct isci_host *, struct isci_port *);

void sci_port_deactivate_phy(
struct isci_port *iport,
struct isci_phy *iphy,
bool do_notify_user);

bool sci_port_link_detected(
struct isci_port *iport,
struct isci_phy *iphy);

enum sci_status sci_port_get_properties(
struct isci_port *iport,
struct sci_port_properties *prop);

enum sci_status sci_port_link_up(struct isci_port *iport,
struct isci_phy *iphy);
enum sci_status sci_port_link_down(struct isci_port *iport,
struct isci_phy *iphy);

struct isci_request;
struct isci_remote_device;
enum sci_status sci_port_start_io(
struct isci_port *iport,
struct isci_remote_device *idev,
struct isci_request *ireq);

enum sci_status sci_port_complete_io(
struct isci_port *iport,
struct isci_remote_device *idev,
struct isci_request *ireq);

enum sas_linkrate sci_port_get_max_allowed_speed(
struct isci_port *iport);

void sci_port_broadcast_change_received(
struct isci_port *iport,
struct isci_phy *iphy);

bool sci_port_is_valid_phy_assignment(
struct isci_port *iport,
u32 phy_index);

void sci_port_get_sas_address(
```

```
struct isci_port *iport,
struct sci_sas_address *sas_address);

void sci_port_get_attached_sas_address(
struct isci_port *iport,
struct sci_sas_address *sas_address);

void sci_port_set_hang_detection_timeout(
struct isci_port *isci_port,
u32 timeout);

void isci_port_formed(struct asd_sas_phy *);
void isci_port_deformed(struct asd_sas_phy *);

int isci_port_perform_hard_reset(struct isci_host *ihost, struct isci_port
*iport,
struct isci_phy *iphy);
int isci_ata_check_ready(struct domain_device *dev);
```

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. Interactive map of Linux kernel [Электронный ресурс]: www.makelinux.net URL:http://www.makelinux.net/kernel_map/
5. Monitoring and Tuning the Linux Networking Stack: Receiving Data [Электронный ресурс] URL: https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/#pci-initialization
6. Операционные системы/Э.Танненбаум, А.Вудхалл — Санкт-Петербург: Питер, 2007
7. Wikipedia Interrupt descriptor table [Электронный ресурс] URL: https://en.wikipedia.org/wiki/Interrupt_descriptor_table
8. Мониторинг и настройка сетевого стека Linux: получение данных [Электронный ресурс] URL: https://habr.com/ru/company/mailru/blog/314168/

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. В процессе выполнения заданий был изучен механизм прерываний.
5. В следствие отсутствия какой-либо информации по поводу портов возникли проблемы с пониманием того какой именно тип порта использовать, как его показывать, а так же поиск кода для написания структуры порта.
6. Много полезной информации нашлось на иностранных ресурсах.