|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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  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

Таблица прерываний (IDT)

IDTR

IRQ handler

CR0

Процессор

Process1

main()

Порт

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

-1

Порт

**@SCI\_PORT\_READY**

Порт готов. Таким образом, пользователь может выполнять операции ввода-вывода на этом порту

CR0

-1

**@SCI\_PORT\_STOPPED**  
Порт успешно остановлен. В этом состоянии новые операции ввода-вывода не разрешены.

**@SCI\_PORT\_SUB\_WAITING**  
Порт запущен и готов, но не имеет активных физ.

CR0

Порт

CR0

Порт

CR0

CR0

CR0

Порт

Порт

Порт

**@SCI\_PORT\_RESETTING**

порт выполняет жесткую  
сброс настроек. Таким образом, пользователь не может выполнять операции ввода-вывода на этом порту.

**@SCI\_PORT\_SUB\_OPERATIONAL**  
\*Происходит либо событие a – передача данных в порт, либо b – получение данных из порта.

**@SCI\_PORT\_SUB\_STOPPING**

порт останавливается. В этом состоянии новые операции ввода-вывода не разрешены, но разрешено завершение существующих операций ввода-вывода

-1

CR0

Порт

**@SCI\_PORT\_SUB\_OPERATIONAL**  
Порт не смог выполнить запрос сброса. В это состояние входит, когда истекает время ожидания запроса сброса порта.

**Сетевой интерфейс в Linux.**

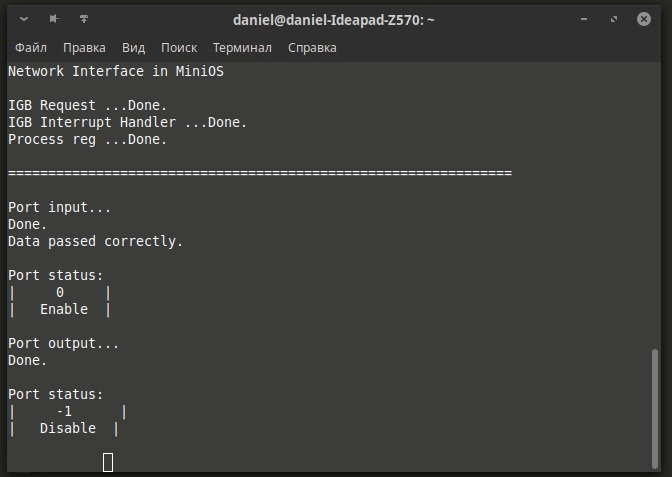
Сетевой интерфейс - это физическое или виртуальное устройство, предназначенное для передачи данных между программами через компьютерную сеть. В данном случае данные передаются между компьютером пользователя и сетью.  
  
Сетевое взаимодействие Linux-компьютера происходит через сетевые интерфейсы. Любые 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 прерываний, возврат к выполнению основного процесса | **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  … |

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/](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.