УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №4

по предмету «Архитектура компьютерной техники и операционных систем»

Вариант 2

Выполнил:

Бражалович А.И.

Проверил:

[Аврамец](https://iis.bsuir.by/employees/d-avramets) Д.В.

Группа 351004

Минск 2024

**Задание**

1. Разработать программу формирования цикла опроса и идентификации устройств PCI, которая будет считывать два первых поля конфигурационного пространства - коды Vendor ID (производитель) и Device ID (устройство).

2. Результатом работы программы должна быть выводимая на экран или в текстовый файл таблица, содержащая следующую информацию:

2.1. адрес устройства (номер шины, номер устройства и номер функции);

2.2. 16-разрядный код производителя (в шестнадцатеричной системе);

2.3. 16-разрядный код устройства (в шестнадцатеричной системе);

2.4. производитель и название устройства.

(2). Если устройство не мост(0-бит поля Header Type =0) вывести и расшифровать значение полей базовых регистров памяти.

(6). Если устройство не мост(0-бит поля Header Type =0) вывести и расшифровать значение поля Interrupt Pin.

(11). Если устройство мост(0-бит поля Header Type =1) вывести и расшифровать значение поля Сlass Code.

**Код программы**

#include <sys/io.h>

#include <stdio.h>

#include "pci\_c\_header.h"

#include <string.h>

#define DATA\_PORT 0x0CFC

#define CONTROL\_PORT 0x0CF8

#define HEADER\_TYPE 0x0C

#define INTERRUPT\_PIN 0x3C

#define BAR 0x10

#define BUS 256

#define DEVICE 32

#define FUNCTION 8

#define NOT\_SPECIFIED "Not specified"

unsigned getAdress(int bus, int device, int function) {

return ((bus << 16) + (device << 11) + (function << 8) + (1 << 31));

}

unsigned getData(const address) {

outl(address, CONTROL\_PORT);

unsigned int reg = inl(DATA\_PORT);

return reg;

}

int isBridge(const int address) {

outl(address, CONTROL\_PORT);

int bridge = inl(DATA\_PORT);

return bridge;

}

void printBARInfo(int address) {

for (int i = 0; i < 6; i++) {

int bar = getData(address + BAR + i \* 4);

printf("BAR%d: 0x%08X ", i, bar);

if ((bar & 0x1) == 0) {

printf("(Memory BAR)\n");

int bar\_addr = ~bar & 0xFFFFFFF0;

int bar\_type = (bar >> 1) & 0x3;

printf("-Base Address: 0x%08X\n", bar\_addr);

switch (bar\_type) {

case 0x0:

printf("--Memory can be mapped to any place in 32-bit address space of system.\n");

break;

case 0x1:

printf("--Less than 1mB.\n");

break;

case 0x2:

printf("--Any place in 64-bit address space of system.\n");

break;

case 0x3:

printf("--Reserved.\n");

break;

default:

printf("--%s", NOT\_SPECIFIED);

}

} else {

printf("(Port BAR)\n");

}

}

}

void printInterruptPinInfo(int address) {

int intPin = (getData(address + INTERRUPT\_PIN) >> 8) & 0xFF;

printf("Interrupt Pin: %d |", intPin);

switch (intPin) {

case 0:

printf("None");

break;

case 1:

printf("INTA#");

break;

case 2:

printf("INTB#");

break;

case 3:

printf("INTC#");

break;

case 4:

printf("INTD#");

break;

case 5:

printf("Reserved");

break;

default:

printf(NOT\_SPECIFIED);

break;

}

}

printCodeClassInfo(const int address) {

int class\_code\_base = (getData(address + 0x08) >> 24) & 0xFF;

switch (class\_code\_base) {

case 0x00:

printf("Classification solver\n");

break;

case 0x01:

printf("Data controller\n");

break;

case 0x02:

printf("Network controller\n");

break;

case 0x03:

printf("Display controller\n");

break;

case 0x04:

printf("Media controller\n");

break;

case 0x05:

printf("Memor controller\n");

break;

case 0x06:

printf("Bridges\n");

break;

case 0x07:

printf("Commutation controller\n");

break;

case 0x08:

printf("System perepheria\n");

break;

case 0x09:

printf("Input Controller\n");

break;

case 0x0A:

printf("Doc-station\n");

break;

case 0x0B:

printf("Processor\n");

break;

case 0x0C:

printf("Bus controller\n");

break;

case 0x0D:

printf("Wireless controller\n");

break;

default:

printf(NOT\_SPECIFIED);

break;;

}

}

void prindVenDevId(const unsigned int reg) {

int ven\_id = reg & 0x0000ffff;

int dev\_id = reg >> 16;

int i;

int exist = 0;

for (i = 0; i < PCI\_DEVTABLE\_LEN; i++) {

if (dev\_id == PciDevTable[i].DevId) {

printf("DeviceId: %d\n", PciDevTable[i].DevId);

printf("Chip: %s\nChip Desc: %s \n", PciDevTable[i].Chip, PciDevTable[i].ChipDesc);

exist = 1;

break;

}

}

if (!exist) {

printf("Chip: None\nChip Desc: None\n");

}

for (i = 0; i < PCI\_VENTABLE\_LEN; i++)

{

if (ven\_id == PciVenTable[i].VenId)

{

printf("VendorId: %d\n", PciVenTable[i].VenId);

printf("Name Full: %s\nName Short: %s \n", PciVenTable[i].VenFull, PciVenTable[i].VenShort);

}

}

}

int main()

{

//printf("Peripherial devs lab3\n\n");

int i,busid,devid;

if(iopl(3)) //задание уровня приоритета

{

printf("I/O Privilege level change error");

return 1;

}//granted privileges 3 for port access

for (int bus\_num = 0; bus\_num < BUS; bus\_num++) {

for (int dev\_num = 0; dev\_num < DEVICE; dev\_num++) {

for (int func\_num = 0; func\_num < FUNCTION; func\_num++) {

unsigned int addr = getAdress(bus\_num, dev\_num, func\_num);

unsigned int reg = getData(addr);

if (reg >> 16 != 0xffff) {

printf("=======================================================================");

printf("\nDevice Address|Bus Number:%d|Device Number:%d|Function Number:%d|\n", bus\_num, dev\_num, func\_num);

prindVenDevId(reg);

if (isBridge((addr + HEADER\_TYPE)) >> 16) {

printf("Bridge\n");

printf("-----------------------------------------------------------------------\n");

printCodeClassInfo(addr);

} else {

printf("Not Bridge\n");

printf("-----------------------------------------------------------------------\n");

printBARInfo(addr);

printf("-----------------------------------------------------------------------\n");

printInterruptPinInfo(addr);

printCodeClassInfo(addr);

}

printf("\n=======================================================================\n");

}

}

}

}

return 0;

}

**Пример работы**

**=======================================================**

**Device Address|Bus Number:2|Device Number:4|Function Number:0|**

**DeviceId: 2016**

**Chip: C73**

**Chip Desc: NVIDIA GeForce 7150m graphics**

**VendorId: 5549**

**Name Full: VMware Inc.**

**Name Short: VMware**

**Not Bridge**

**-----------------------------------------------------------------------**

**BAR0: 0x00000000 (Memory BAR)**

**-Base Address: 0xFFFFFFF0**

**--Memory can be mapped to any place in 32-bit address space of system.**

**BAR1: 0x00000000 (Memory BAR)**

**-Base Address: 0xFFFFFFF0**

**--Memory can be mapped to any place in 32-bit address space of system.**

**BAR2: 0x00000000 (Memory BAR)**

**-Base Address: 0xFFFFFFF0**

**--Memory can be mapped to any place in 32-bit address space of system.**

**BAR3: 0x00000000 (Memory BAR)**

**-Base Address: 0xFFFFFFF0**

**--Memory can be mapped to any place in 32-bit address space of system.**

**BAR4: 0x00000000 (Memory BAR)**

**-Base Address: 0xFFFFFFF0**

**--Memory can be mapped to any place in 32-bit address space of system.**

**BAR5: 0xFD5EE000 (Memory BAR)**

**-Base Address: 0x02A11FF0**

**--Memory can be mapped to any place in 32-bit address space of system.**

**-----------------------------------------------------------------------**

**Interrupt Pin: 1 |INTA#**

**=======================================================**