Учреждение образования

«Брестский государственный технический университет»

Факультет электронно-информационных систем

Кафедра «ЭВМ и системы»

СОГЛАСОВАНО

Заведующий кафедрой

«ЭВМ и системы»

_____ С.С. Дереченник

«20» декабря 2023 г.

СОГЛАСОВАНО

Декан факультета электронно-

информационных систем

_____ А.Н. Парфиевич

«20» 12 2023 г.

ЭЛЕКТРОННЫЙ УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ «АРХИТЕКТУРА ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ»

для специальности:

1-40 02 01 Вычислительные машины, системы и сети

Составитель: М.В. Николаюк-Ртищева, старший преподаватель кафедры «ЭВМ и системы».

Рассмотрено и утверждено на заседании Научно-методического совета университета 21.12.2023 г., протокол № 2.

№ регистрации УМК 23/24-06

ПЕРЕЧЕНЬ МАТЕРИАЛОВ В КОМПЛЕКСЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Широкое распространение электронно-вычислительных машин для решения различных производственных задач и задач управления технологическими процессами требует от инженера-системотехника знаний особенностей построения и принципов работы современных персональных компьютеров (ПК) и их отдельных компонентов, чтобы иметь возможность использовать общие особенности функционирования вычислительных машин при решении конкретных задач. Целью изучения дисциплины «Архитектура персональных компьютеров» является изучение организации и функционирования аппаратуры персональных ЭВМ, а также приобретение знаний и навыков в области низкоуровневого программирования устройств современных компьютеров.

Электронный учебно-методический комплекс (ЭУМК) по дисциплине «Архитектура персональных компьютеров» представляет собой комплекс систематизированных учебных и методических материалов и предназначен для подготовки студентов специальности 1-40 02 01 Вычислительные машины, системы и сети.

ЭУМК разработан в соответствии со следующими нормативными документами:

– Положение об учебно-методическом комплексе на уровне высшего образования, утвержденное постановлением Министерства образования Республики Беларусь № 427 от 25.11.2022;

– Положение об учебно-методическом комплексе по учебной дисциплине учреждения образования «Брестский государственный технический университет» № 8 от 13.01.2023;

– Учебная программа учреждения высшего образования по учебной дисциплине «Архитектура персональных компьютеров», утвержденная 29.06.2022, регистрационный № УД-22-1-165/уч.

Цели ЭУМК:

– обеспечение качественного методического сопровождения процесса обучения;

– организация эффективной самостоятельной работы студентов.

Содержание и объем ЭУМК полностью соответствуют образовательному стандарту высшего образования специальности 1-40 02 01 Вычислительные машины системы и сети, а также учебно-программной документации

образовательных программ высшего образования. Материал представлен на требуемом методическом уровне и адаптирован к современным образовательным технологиям.

Структура ЭУМК:

1. Теоретический раздел ЭУМК представлен опорным конспектом лекций, включающим план каждой лекции со ссылками на необходимые страницы литературных источников и краткое содержание лекционного материала по всем темам.
2. Практический раздел ЭУМК содержит материалы для проведения лабораторных работ. Методические материалы к выполнению восьми лабораторных работ представлены в ЭУМК в электронном виде, бумажный экземпляр методических указаний находится в учебной лаборатории кафедры.
3. Раздел контроля знаний ЭУМК содержит перечень вопросов для подготовки к экзамену, образец билета на экзамен.
4. Вспомогательный раздел включает учебную программу по дисциплине «Архитектура персональных компьютеров» и информационно-методическую часть (список основной и дополнительной литературы).

Рекомендации по организации работы с ЭУМК:

Использование разработанного ЭУМК предполагает работу студентов с конспектом лекций при подготовке к выполнению и защите лабораторных работ, к сдаче экзамена по дисциплине. Изучение дисциплины на основе ЭУМК предполагает продуктивную учебную деятельность, позволяющую сформировать профессиональные компетенции будущих специалистов, обеспечить развитие познавательных и созидательных способностей личности. ЭУМК направлен на повышение эффективности учебного процесса. ЭУМК способствует успешному усвоению студентами учебного материала, дает возможность планировать и осуществлять самостоятельную работу студентов, обеспечивает рациональное распределение учебного времени по темам учебной дисциплины и совершенствование методики проведения занятий. Для работы с ЭУМК необходим IBM-PC-совместимый ПК стандартной конфигурации.

1. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

1.1 КОНСПЕКТ ЛЕКЦИЯ ПО ДИСЦИПЛИНЕ «АРХИТЕКТУРА ПК»

Тема 1 История развития вычислительных систем

- поколения ЭВМ;
- архитектура фон-Неймана;
- гарвардская архитектура.

Литература [1, стр. 20-42, 118-124]

По этапам создания и используемой элементной базе ЭВМ условно можно разделить на следующие поколения:

- 1-е поколение, 50-е гг.: ЭВМ на электронных вакуумных лампах. Отличались большими габаритами, большим потреблением энергии, малым быстродействием, низкой надежностью, программированием в кодах.
- 2-е поколение, с конца 50-х гг.: ЭВМ на транзисторах. Улучшены показатели производительности, энергопотребления, надежности. Для программирования используются алгоритмические языки.
- 3-е поколение, начало 60-х гг.: ЭВМ на интегральных схемах с малой степенью интеграции (тысячи транзисторов и диодов в одном корпусе). Резкое снижение габаритов ЭВМ, повышение их надежности, производительности.
- 4-е поколение, с середины 70-х гг.: ЭВМ на больших и сверхбольших интегральных схемах – микропроцессорах (миллионы транзисторов в одном кристалле). Улучшились все технические характеристики. Массовый выпуск ПК.
- 5-е поколение с середины 80-х гг.: ЭВМ со множеством параллельно работающих микропроцессоров. В настоящее время ведутся интенсивные работы по созданию оптоэлектронных ЭВМ с массовым параллелизмом и нейронной структурой, представляющих собой распределенную сеть большого числа (десятки тысяч) микропроцессоров, моделирующих архитектуру нейронных систем.

Первой работающей машиной первого поколения стала Манчестерская малая экспериментальная машина, созданная в Манчестерском университете в 1948 году, одним из разработчиков которой был Джон фон Нейман. Принципы, по которым создана эта машина, составляют суть т.н. архитектуры фон Неймана, по которой построено большинство современных ПК, и заключаются в

следующем:

- Использование двоичной системы счисления. Для представления данных и команд используется двоичная система счисления. Устройства можно делать достаточно простыми, арифметическо-логические команды выполняются быстро.

- Принцип однородности памяти. Память компьютера используется не только для хранения данных, но и программ. При этом их способ записи одинаков. Над командами можно выполнять такие же действия, что и над данными.

- Принцип адресности. Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы. В любой момент можно обратиться к любой ячейке памяти по ее адресу.

- Принцип программного управления. Работа ЭВМ контролируется программой, состоящей из набора команд. ЭВМ можно запрограммировать.

- Принцип условного перехода. В программах можно реализовать возможность перехода к любому участку кода.

Машина фон Неймана состоит из арифметико-логического устройства - АЛУ, устройства управления – УУ, запоминающего устройства - ОЗУ, а также устройств ввода/вывода и внешнего запоминающего устройства (см. рисунок 1).

Рисунок 1 – Структура ЭВМ по фон Нейману

В качестве недостатка архитектуры фон Неймана можно назвать возможность непреднамеренного нарушения работоспособности системы (программные ошибки) и преднамеренное уничтожение ее работы (вирусные атаки).

В гарвардской архитектуре память команд и память данных физически разделены. Шины, соединяющие их с процессором, могут иметь как разную разрядность, так и разный объем памяти. Это дает выигрыш в быстродействии, поскольку за один машинный цикл процессор может получить и команды, и данные. Основное применение — в микроконтроллерах, от которых необходимо максимальное быстродействие при заданной тактовой частоте. Недостатком является сложность технической реализации двух независимых шин, требующая дополнительных аппаратных затрат. Структура системы с гарвардской архитектурой показана на рисунке 2.

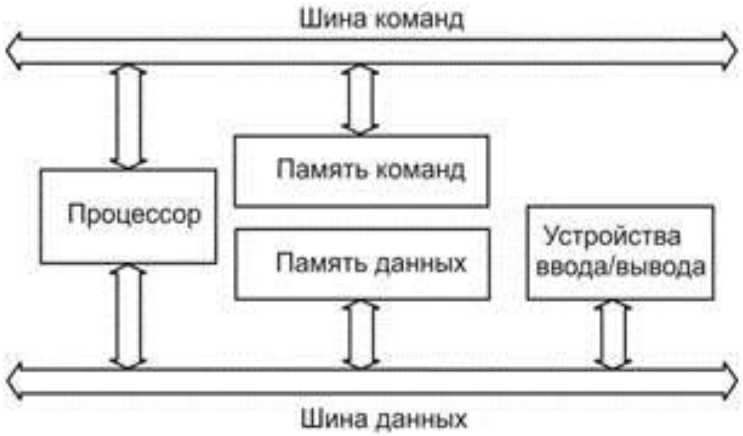

Рисунок 2 – Структура ЭВМ с гарвардской архитектурой

Тема 2 Архитектура вычислительных систем

- понятие архитектуры ПК;
- классификация Флинна;
- основные узлы вычислительной системы;
- память: ОЗУ, ПЗУ, кэш-память;
- архитектура микропроцессора;
- устройства ввода-вывода;
- шины;
- контроллеры и адаптеры.

Литература [1, стр. 118-124, 463], [2, стр. 91-107]

Основная компоновка частей компьютера и связь между ними называется архитектурой. При этом определяется состав входящих в него компонент, принципы их взаимодействия, а также их функции и характеристики.

Попытки систематизировать множество архитектур вычислительных систем предпринимались достаточно давно. Самой известной классификацией является классификация М. Флинна. В ее основу положено понятие потока, под которым понимается последовательность элементов, команд или данных, обрабатываемая процессором. В зависимости от количества потоков команд и потоков данных Флинн выделяет четыре класса архитектур: SISD, MISD, SIMD, MIMD.

SISD (Single Instruction Single Data) – "один поток команд, один поток данных" (ОКОД) – исполнение одним процессором одного потока команд, который обрабатывает данные, хранящиеся в одной памяти.

SIMD (Multiple Data stream processing) –"один поток команд, много потоков данных"(ОКМД)– исполнение одной команды несколькими процессорами. Команда выбирается из памяти центральным контроллером SIMD-системы, но выполняется отдельными процессорами над своими локальными данными.

MISD (Multiple Instruction Single Data) –"много потоков команд, один поток данных"(МКОД) – данные подаются на набор процессоров, каждый из которых исполняет свою программу их обработки.

MIMD (Multiple Instructions - Multiple Data) –"много потоков команд, много потоков данных"(МКМД) – набор процессоров независимо выполняет различные наборы команд, обрабатывающих различные наборы данных.

В основу архитектуры современных ПК положен магистрально-модульный принцип (смотри рисунок 3). Модульный принцип позволяет потребителю самому комплектовать нужную конфигурацию и производить ее модернизацию. Модульная организация компьютера опирается на магистральный (шинный) принцип обмена информацией между устройствами по одним и тем же шинам поочередно.

Рисунок 3 – Структура ПК

Системная шина включает в себя три многоразрядные шины: шину данных, шину адреса и шину управления. Шина управления (ШУ) предназначена для передачи управляющих импульсов и импульсов синхронизации сигналов ко всем устройствам ПК. Шина адреса (ША) предназначена для передачи кода адреса ячейки памяти или порта ввода/вывода внешнего устройства. Шина данных (ШД) предназначена для передачи разрядов цифрового кода. К магистрали подключаются процессор и оперативная память, а также устройства ввода/вывода и хранения информации, которые обмениваются информацией на машинном языке (последовательностями нулей и единиц в форме электрических импульсов).

Главным модулем системы является процессор, который выполняет команды программы, организует обращение к памяти, инициирует работу устройств. МП имеет сложную структуру, в качестве его компонент можно выделить:

- АЛУ для выполнения операций над целочисленными данными;
- блок FPU для операций над числами с плавающей точкой;
- регистры – сверхоперативная память, работающая со скоростью процессора;
- устройство управления, управляющее работой всех узлов МП посредством выработки и передачи другим его компонентам управляющих импульсов;
- контроллер прерываний, обслуживающий запросы от устройств при возникновении в них какого-либо события.

Память – устройство для хранения информации в виде данных и программ. Память делится на внутреннюю (расположенную на системной плате) и внешнюю (размещенную на разнообразных внешних носителях информации).

Внутренняя память в свою очередь подразделяется на:

- ПЗУ, содержащее информацию, сохраняемую при отключенном питании, служит для хранения программ тестирования оборудования, начальной загрузки ПК.

- ОЗУ, предназначенную для хранения программ и данных, сохраняемых на период работы ПК; энергозависима, при отключении питания информация теряется.

- Кэш-память – для временного хранения промежуточных результатов и содержимого часто используемых ячеек ОЗУ и регистров МП; малое время доступа.

Внешняя память используется для долговременного хранения информации. Простым примером являются накопители на жестких магнитных дисках (НЖМД).

Мониторы – устройства отображения информации – по принципу построения можно разделить на ЭЛТ-мониторы (устаревшие) и жидкокристаллические.

Периферийные устройства (мышь, клавиатура, сканер, принтер) организуют ввод-вывод информации в систему.

Все блоки через унифицированные разъемы подключаются к шине непосредственно или через контроллеры (адаптеры). Адаптер – это более сложное устройство, имеющее внутреннюю оперативную память.

Доступ к контроллерам и адаптерам осуществляется через порты ввода-вывода, которые представляют собой адрес устройства в адресном пространстве ПК.

Тема 3 Базовая архитектура процессора

- архитектура, характеристики и функции микропроцессоров x86 серии;
- обзор уровня архитектуры команд;
- модель микропроцессора для программиста.

Литература [1, стр. 434-443], [2, стр. 76-84, 379-388]

Микропроцессор (CPU) – функционально-законченное программно-управляемое устройство, предназначенное для обработки информации под управлением программы, находящейся сейчас в оперативной памяти.

Архитектура микропроцессора – принцип его внутренней организации, общая структура, конкретная логическая структура отдельных устройств.

Микроархитектура процессора – аппаратная организация и логическая структура процессора, регистры, управляющие схемы и связывающие их магистрали.

Макроархитектура микропроцессора – это система команд, типы обрабатываемых данных, режимы адресации и принципы работы микропроцессора.

Для одной и той же архитектуры применяются различные микроархитектурные реализации. В соответствии с архитектурными особенностями различают:

Микропроцессоры с RISC архитектурой – процессоры с сокращенной системой команд – имеют набор однородных регистров универсального назначения, простую систему команд, коды инструкций с фиксированной длиной. Выполняются инструкции за минимальное число тактов синхронизации.

Микропроцессоры с CISC архитектурой – процессоры с полным набором инструкций (семейство x86) – имеют неоднородный состав регистров, широкий набор команд. Усложняется декодирование инструкций, возрастает число тактов при их выполнении. Начиная с 486-ых, CISC-процессор имеет RISC-ядро.

Микропроцессоры с MISC архитектурой – процессоры с минимальной системой команд – состоят из RISC CPU (для основных команд) и ПЗУ микропрограммного управления (содержит программы для команд расширения).

Архитектура VLIW – архитектура процессоров с несколькими вычислительными устройствами, выполняющими операции параллельно. Аналог CISC со спекулятивным исполнением команд на этапе компиляции.

Кроме типа архитектуры, основными характеристиками процессора являются:

- Тактовая частота – количество тактов, производимых процессором за 1 секунду. Современные микропроцессоры работают на частотах до 4,2 ГГц.
- Разрядность – максимальное количество разрядов двоичного кода, которые могут передаваться одновременно.

- Объем адресуемой памяти – максимальный объем памяти, который обслужит процессор. Вычисляется из разрядности шины адреса.
- Частота системной шины. Системная шина служит для связи микропроцессора с остальными устройствами. Сегодня 2-8 ГГц.
- Объем Кэш-памяти. Кэш-память – это быстрая память малой емкости, используемая процессором для ускорения операций с ОЗУ. Микропроцессоры (Skylake, например) имеют L1 объемом 64КБ, L2 – 256 Кб, L3 – 8 Мб.
- Технологический процесс производства определяет размеры элементов и соединений между ними в интегральной схеме (14 нм, 10 нм).

Основные функции микропроцессора:

- выборка команд из ОЗУ (то есть чтение команд из основной памяти);
- декодирование команд (определение назначения команды, способа исполнения и адресов операндов);
- выполнение операций, закодированных в командах;
- управление пересылкой информации между регистрами, ОЗУ и ПУ;
- обработка внутрипроцессорных и программных прерываний;
- управление различными устройствами, входящими в состав компьютера.

Основным элементом микропроцессора является ядро, содержащее основные функциональные блоки и выполняющее один поток команд. Сегодня процессоры многоядерные, то есть выполняют одновременно несколько потоков команд. В состав микропроцессора входят различные устройства (смотри рисунок 4). АЛУ предназначено для выполнения арифметических и логических операций. Устройство управления УУ координирует взаимодействие частей ПК, управляет работой АЛУ и регистров, формирует адреса ячеек памяти для выполняемой операции, получает сигналы от генератора тактовых импульсов. Микропроцессорная память (регистры) предназначена для кратковременного хранения, записи и выдачи информации. Интерфейсная система предназначена для связи с другими устройствами компьютера.

В состав микропроцессорного блока входят также дополнительные платы с интегральными микросхемами, расширяющие возможности микропроцессора (математический сопроцессор, контроллер прямого доступа к памяти, сопроцессор ввода-вывода, контроллер прерываний, кэш-память, северный мост и др.).

Рисунок 4 – Общая структура процессора

Уровень архитектуры набора команд — это тот уровень, на котором компьютер представляется программисту, пишущему программы на машинном языке (или то, что получается в результате работы компилятора). Чтобы получить программу уровня архитектуры набора команд, создатель компилятора должен знать, какая модель памяти используется в машине, какие регистры, типы данных и команды имеются в наличии и т. д. Вся эта информация в совокупности и определяет уровень архитектуры набора команд.

Таким образом, для программиста процессор состоит из набора регистров общего и специального назначения. Выделяют РОН ax (ax=16 бит, eax=32бита, rax=64бита), bx (ebx, rbx), cx (ecx, rcx), dx (edx, rdx), si и di (esi и edi, rsi и rdi), сегментные регистры кода cs, данных ds, дополнительные es, gs, fs. Также есть регистры стека: стек (ss), указатель стека sp, указатель базы кадра стека (bp, ebp, rbp). Регистр указателя команд (ip, eip, rip) содержит смещение следующей подлежащей выполнению команды. Регистр состояния (FLAGS) хранит сведения о текущих режимах работы.

Тема 4 История развития процессоров Intel

- превое поколение процессоров;
- процессоры Pentium;
- процессоры Intel Core.

Литература [7, стр. 86-100], [10], [11]

Первое поколение (процессоры i8086 и i8088) задало архитектурную основу – набор неравноправных 16-разрядных регистров, сегментную систему адресации памяти в пределах 1 Мбайт с большим разнообразием режимов, систему команд, систему прерываний. В процессорах применялась "малая"

конвейеризация - пока одни узлы выполняли текущую инструкцию, блок предварительной выборки выбирал из памяти следующую. На выполнение каждой инструкции уходило в среднем по 12 тактов процессорного ядра. Второе поколение (i80286 с сопроцессором i80287) привнесло в семейство защищенный режим, позволяющий использовать виртуальную память размером до 1 Гбайт для каждой задачи. Защищенный режим является основой для построения многозадачных операционных систем, но не нашел массового применения. На выполнение инструкции уходило по 4,5 такта. Третье поколение (i386/387) ознаменовалось переходом к 32-разрядной архитектуре IA-32. Увеличился объем адресуемой памяти (до 4 Гбайт реальной, 64 Тбайт виртуальной). Частота достигла 40 МГц. В четвертом поколении (i486) значительно усложнен исполнительный конвейер – основные операции выполняет RISC-ядро, "задания" для которого готовят из входных CISC-инструкций x86. Производительность конвейера процессора оторвалась от возможностей доставки инструкций и данных из оперативной памяти, и прямо в процессор ввели быстродействующий первичный кэш объемом 8-16 Кбайт. В этом же поколении отказались от внешнего сопроцессора: теперь он размещается на одном кристалле с центральным и называется FPU. Пятое поколение – процессор Pentium у Intel и K5 у AMD – привнесли суперскалярную архитектуру. Суперскалярность означает наличие более одного конвейера. У процессоров пятого поколения после блоков предварительной выборки и первой стадии декодирования инструкций имеется два конвейера. Конвейеризирован и блок FPU. Процессор с такой архитектурой может одновременно "выпускать" до двух выполненных инструкций за такт. В процессорах применяется блок предсказания ветвлений. Позже появилось расширение MMX (принцип SIMD): одна инструкция выполняет действия сразу над несколькими (2, 4 или 8) комплектами операндов. Особенностью шестого поколения (процессоры Intel Pentium Pro, Pentium II, Pentium III, Celeron и Xeon) является динамическое исполнение инструкций, использование расширений SSE (Intel) и 3DNow! (AMD) для чисел с плавающей точкой. Число тактов на инструкцию у Pentium Pro сократилось до 0,5 такта.

В 2006-ом году были представлены первые процессоры Intel Core 2, одной из особенностей которых стала поддержка 64-бит. Были доступны процессоры с разным количеством ядер Core 2 Solo, Core 2 Duo, Core 2 Quad, Core 2 Extreme (модели с повышенными тактовыми частотами). В 2009 году Intel меняет наименование линеек процессоров: производительные решения

получили наименование Core i3, i5, i7. Первое поколение Intel Core было построено по 45 нм, а после и по 32 нм техпроцессу в соответствии с экстенсивной стратегией разработки «Тик-так». Цикл разработки делится на две стадии «тик» и «так». «Тик» означает уменьшение технологического процесса на основе существующей микроархитектуры. «Так» означает выпуск микропроцессоров с новой микроархитектурой на основе существующего технологического процесса. Позже Intel перешла на трехшаговый график обновления модельного ряда — «Тик-Так-Ток». Дополнительный третий шаг подразумевает доработку уже имеющегося техпроцесса и архитектуры и, как результат, существенное повышение частоты. Последними процессорами Intel являются процессоры десятого поколения под кодовым семейством Ice Lake, изготовленными по технологии 10нм и имеющими до 10 ядер в зависимости от модели; основной упор сделан на увеличение многопоточности, повышенное энергосбережение и использование искусственного интеллекта.

Тема 5 Расширение архитектуры процессора

- математический сопроцессор;
- технология MMX;
- регистры MMX/XMM, команды;
- технология SSE.

Литература [7, стр. 84, 85], [8], [9]

Intel уже очень давно использует практику создания специализированных наборов команд в процессорах Intel для повышения производительности специфических ресурсоемких приложений. Одним из первых примеров подобного рода разработок был набор команд для вычислений с плавающей запятой, впервые реализованный в процессоре 8086 в 1978 году (математический сопроцессор). С появлением процессоров пятого поколения (Pentium, Pentium MMX) в программной модели процессора Intel следует различать два слоя архитектуры: базовый и модельно-зависимый.

Слой базовой архитектуры включает в себя элементы архитектуры, поддержку и неизменность, которых производитель гарантирует. Это набор и структура основных устройств процессора, системных регистров и регистров общего назначения и т. д.

Модельно-зависимый слой архитектуры включает в себя средства, поддержка которых привязана к конкретной модели процессора. MMX-расширение процессора предназначено для поддержки приложений, ориентированных на работу с большими массивами данных целого и вещественного типов, над которыми выполняются одинаковые операции. С данными такого типа обычно работают мультимедийные, графические, коммуникационные программы. Именно по этой причине данное расширение архитектуры процессоров Intel и названо MMX (MultiMedia eXtensions — мультимедийные расширения).

После появления процессора Pentium III следует различать MMX-расширения двух типов — целочисленное и с плавающей запятой. Каждое из этих MMX-расширений имеет свою программную модель и не зависит от другого. Различают целочисленное MMX-расширение и MMX-расширение с плавающей запятой – XMM-расширение.

MMX-команды пересылки, подобно их целочисленным аналогам, являются наиболее часто используемыми. Эти команды осуществляют передачу информации в MMX-регистры и из регистров. MMX-команды пересылки работают с 32- и 64-раэрядными операндами. В группу арифметических входят команды для реализации трех основных арифметических операций: сложения, вычитания, умножения. Команды логических операций предназначены для поразрядной обработки содержимого двух MMX-рсгистров или MMX-регистра и 64-разрядного операнда в памяти. Эти команды реализуют логические операции И, И-НЕ, ИЛИ.

В микроархитектуре P6 Intel впервые представила набор команд Streaming SIMD Extensions (SSE), реализованный в процессоре Pentium III. Набор команд SSE расширял возможности MMX и позволял одновременно выполнять команды SIMD над четырьмя упакованными элементами данных с плавающей запятой одинарной точности.

На базе микроархитектуры NetBurst (процессор Pentium 4) был разработан набор команд SSE2, который стал расширением SSE (и MMX). SSE2 предназначался для повышения степени параллелизма при выполнении команд MMX и SSE. Поддерживалась обработка 128-разрядных целочисленных данных и упакованных данных с плавающей запятой двойной точности. В целом, набор команд SSE2 содержал 144 дополнительные инструкции, которые обеспечивали повышение производительности приложений.

В процессоре Pentium 4 был реализован набор команд SSE3. Он включал 13 дополнительных инструкций SIMD. SSE4 – самое масштабное и значительное расширение архитектуры Intel ISA. Набор команд SSE4 включает множество новых инновационных инструкций, которые можно разделить на две основные категории: векторизирующий компилятор (мультимедиа-ускорители) и ускорители обработки строк и текстовой информации. Новые инструкции дополняют обширную архитектуру набора команд (instruction set architecture, ISA) Intel 64.

Команды обработки строк и текста позволяют увеличить производительность приложений для обработки данных, поиска и других текстовых приложений. Они включают в себя набор команд сравнения упакованных строк, который позволяет производить несколько операций сравнения и поиска за одну инструкцию.

Чтобы получить максимальную выгоду от использования новых инструкций, старые приложения необходимо перекомпилировать с помощью обновленных версий компиляторов Intel или других производителей. В специализированных мультимедиа-приложениях рост производительности может быть очень значительным.

Тема 6 Архитектура многоядерного процессора

- состав и принцип работы ядра современного процессора;
- структура современного многоядерного процессора.

Литература [1, стр. 446-448], [2, стр. 84-94], [7, стр. 78-82]

Современный процессор – это сложное и высокотехнологическое устройство, включающее в себя последние достижения в области вычислительной техники и других областей науки.

Упрощенная структурная схема современного многоядерного процессора представлена на рисунке 5.

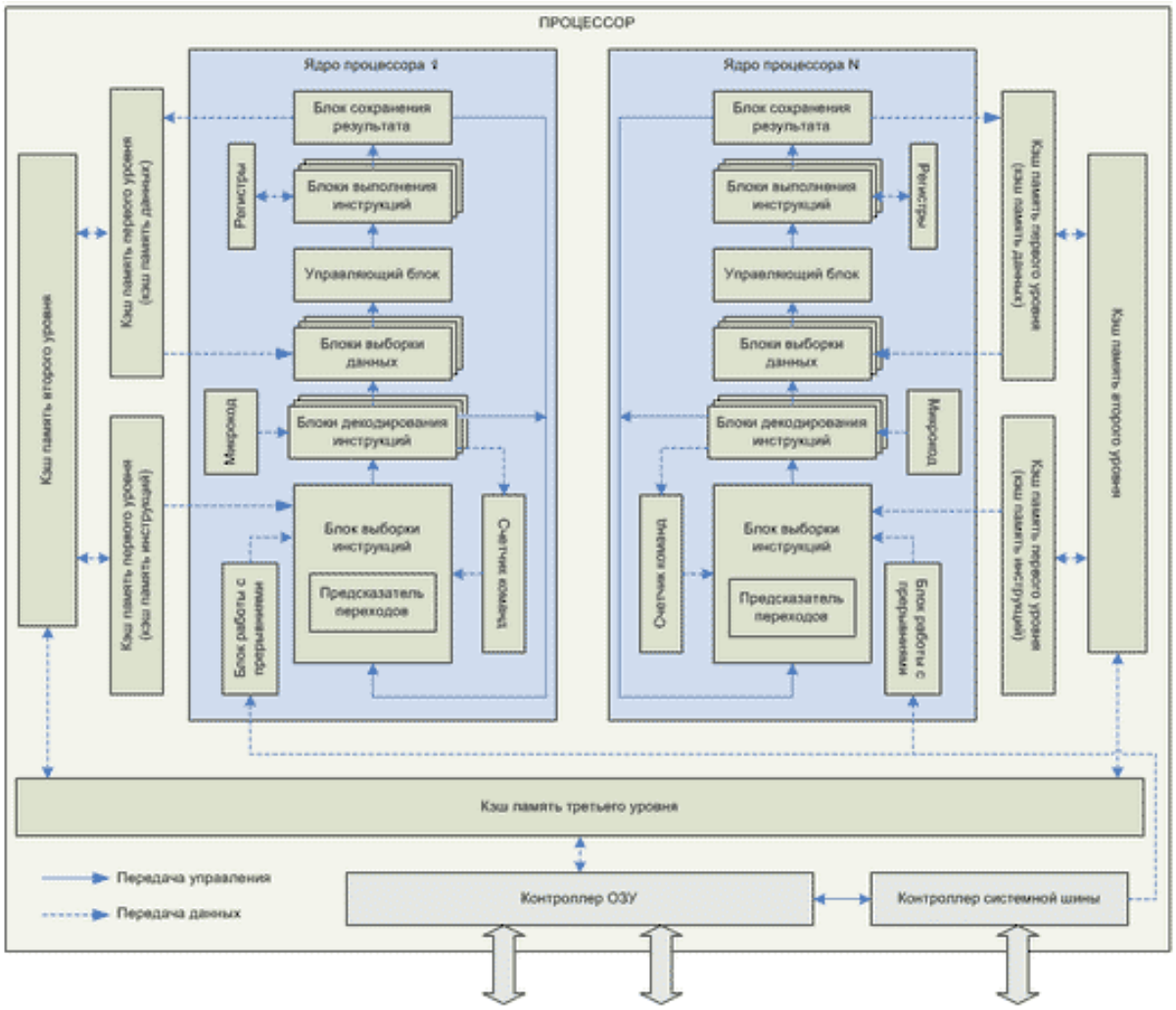

Рисунок 5 – Структура многоядерного процессора

Большинство современных процессоров состоит из:

- нескольких ядер, осуществляющих выполнение всех инструкций;
- трех уровней КЭШ-памяти, ускоряющих взаимодействие процессора с ОЗУ;
- контроллера ОЗУ;
- контроллера системной шины (DMI, QPI, HT и т.д.);
- северного моста и графического адаптера (опционально).

Многоядерный процессор характеризуется следующими параметрами: типом микроархитектуры, тактовой частотой, набором выполняемых команд, количеством уровней КЭШ-памяти и их объемом, типом и скоростью системной шины, размерами обрабатываемых слов, наличием или отсутствием встроенного контроллера памяти, типом поддерживаемой оперативной памяти, объемом адресуемой памяти, наличием или отсутствием встроенного графического ядра, энергопотреблением.

Тема 7 Способы повышения производительности ядра процессора

- конвейеризация;
- суперскалярность;
- параллельная обработка данных;
- технологии энергосбережения;
- кэш-память.

Литература [2, стр. 84-94], [1, стр. 83-86]

Увеличение производительности ядра процессора, за счет поднятия тактовый частоты, имеет жесткое ограничение. Увеличение тактовой частоты влечет за собой повышение температуры процессора, энергопотребления и снижение стабильности его работы и срока службы. Поэтому разработчики процессоров применяют различные архитектурные решения, позволяющие увеличить производительность процессоров без увеличения тактовой частоты.

Рассмотрим основные способы повышения производительности процессоров.

- Конвейеризация. Для уменьшения простоя блоков процессора выполнение инструкций выполняется по конвейеру: по мере освобождения блоков ядра они загружаются обработкой следующей инструкции, не дожидаясь завершения предыдущей.
- Суперскалярность – архитектура вычислительного ядра, при которой наиболее нагруженные блоки могут входить в нескольких экземплярах. В этом случае блоки, работающие дольше, не будут задерживать весь конвейер.
- Параллельная обработка данных. Подавляющее большинство современных процессоров имеют два и более ядра. Практически получается несколько процессоров, способных независимо решать каждый свои задачи.
- Технология Hyper-Threading (Intel) позволяет каждому ядру процессора выполнять две задачи одновременно, по сути, делая из одного реального ядра два виртуальных. При этом сохраняется состояние сразу двух потоков, так как у ядра есть свой набор регистров, свой счетчик команд и свой блок работы с прерываниями для каждого потока. В результате, операционная система видит такое ядро как два.

- Технология Turbo Boost. Производительность большинства современных процессоров в домашних условиях можно немного разогнать – заставить работать на частотах, превышающих номинальную, т.е. заявленную производителем.
- Технологии снижения энергопотребления процессора. Технология EIST у Intel и Cool'n'Quiet у AMD позволяют динамически изменять энергопотребление процессора за счет изменения тактовой частоты процессора и напряжения. В случаях, когда процессор используется не полностью, его тактовую частоту можно снизить, уменьшая коэффициент умножения.
- КЭШ-память призвана сократить время выборки команд и данных из оперативной памяти и выполняет роль промежуточного буфера с быстрым доступом. КЭШ-память строится на базе дорогой SRAM-памяти, обеспечивающей доступ к ячейкам памяти гораздо более быстрый, чем к ячейкам DRAM-памяти. К тому же SRAM-память не требует постоянной регенерации, что также увеличивает ее быстродействие.

Тема 8 Материнская плата

- основные компоненты материнской платы состав;
- чипсет: северный и южный мост;
- форм-фактор материнской платы; подключение устройств к плате.

Литература [3, стр. 60-110]

Материнская плата – сложная многослойная печатная плата, выступающая интегратором для всех других устройств, определяет функциональные возможности компьютера. Структурная схема материнской платы представлена на рисунке 6.

Основным компонентом материнской платы является набор микросхем – чипсет, состоящий из микросхем северного и южного моста.

Северный мост или контроллер-концентратор памяти, обеспечивающий взаимодействие процессора с памятью и видеоадаптером, определяет параметры (тип, частоту, пропускную способность) системной шины, процессора, оперативной памяти и подключенного видеоадаптера. В современных системах функция северного моста полностью перенесена в центральный процессор.

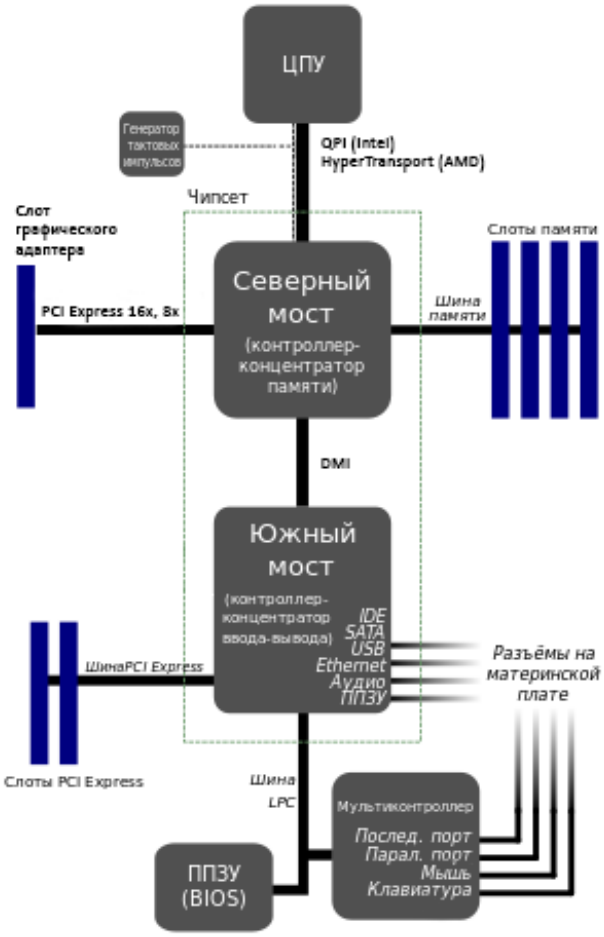

Рисунок 6 – Структура материнской платы

Южный мост или контроллер-концентратор ввода-вывода включает в себя контроллеры шин PCI Express, Super I/O, DMA контроллер, контроллер прерываний, RTC, ПЗУ, SATA-, USB-контроллеры, звуковой контроллер, Ethernet-контроллер.

Форм-фактор материнской платы — стандарт, определяющий размеры материнской платы, расположение интерфейсов шин, портов ввода-вывода, разъёма процессора, слотов для ОЗУ, а также тип разъема для подключения блока питания. Устаревшими являются форматы Baby-AT, AT, LPX, Mini-ATX. Современные форматы: ATX, Micro-ATX, NLX, WTX, Mini-, Nano-ITX, BTX, Micro-, Pico-BTX.

Для подключения устройств к материнской плате имеются специальные разъемы, которые могут располагаться как внутри корпуса компьютера, так и снаружи на задней и передней части системного блока.

Тема 9 Системные устройства

- тактовый генератор;
- часы реального времени;
- системный таймер;
- устройства для отсчета времени в ПК;
- управление питанием и энергопотреблением;
- подсистема ACPI.

Литература [4, стр. 162-170]

Кроме микропроцессора на материнской плате располагаются и другие важные модули, обеспечивающие работоспособность компьютера: контроллеры прерываний, контроллеры прямого доступа, тактовый генератор, часы реального времени, системный таймер, контроллер шины. Они входят в состав южного моста.

Системный тактовый генератор генерирует сигналы синхронизации для работы микропроцессора, всех контроллеров и системной шины. Для обеспечения высокой стабильности тактовых частот и их независимости от температуры применяются кварцевые резонаторы. В компьютерах на базе процессоров x86 применяется деление опорной тактовой частоты для синхронизации системной шины и внутреннее умножение частоты в процессорах.

Часы реального времени – электронная схема для учёта хронометрических данных (текущее время, дата, день недели и др.), представляет собой систему из автономного источника питания и учитывающего устройства. Точность отсчета времени зависит от параметров задающего кварцевого резонатора, который формирует тактовые импульсы с частотой 32,768 кГц. Секундные, минутные и т.д. импульсы формируются с помощью делителей частоты (см. рисунок 7).

Значение часов реального времени содержится в ПЗУ в 14 первых ячейках КМОП-памяти. Для работы с RTC также можно использовать порты 70h и 71h. RTC вырабатывают аппаратное прерывание IRQ8.

Системный таймер – это устройство, называемое также программируемым интервальным таймером (PIT), подключенное к линии запроса на прерывание IRQ0, вырабатывает прерывание INT 8h приблизительно 18,2 раза в секунду. Таймер обычно реализуется на микросхеме i8254 и состоит из трех независимых каналов.

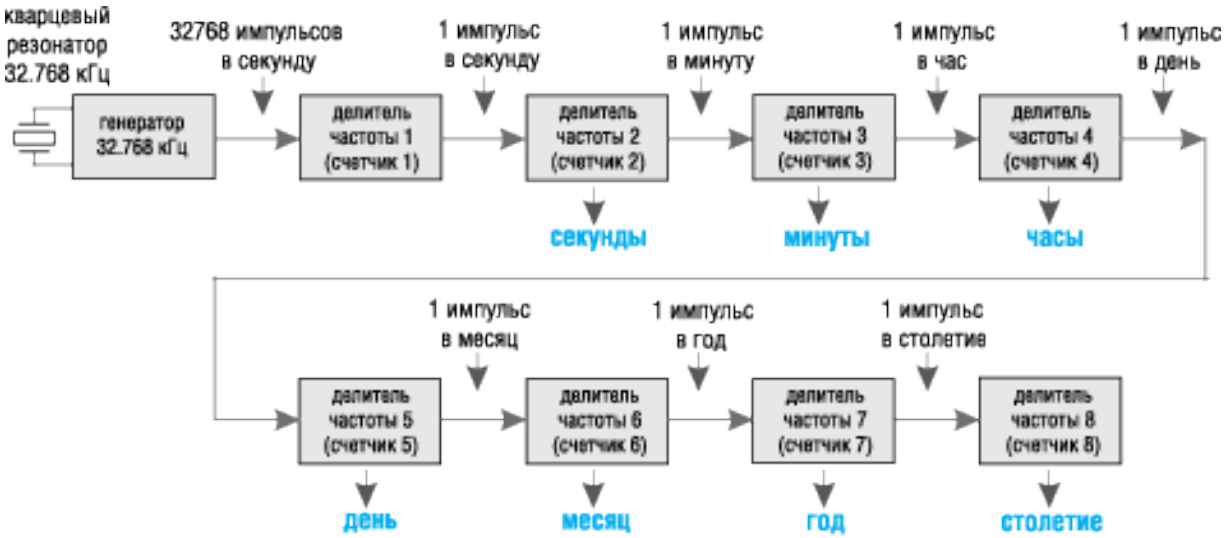

Рисунок 7 – Базовая структура часов реального времени

Канал 0 используется как генератор импульсов с частотой примерно 18.2 Гц. Именно эти импульсы вызывают аппаратное прерывание INT 8h. Канал 1 используется для генерации запроса к контроллеру DMA и регенерации содержимого динамической памяти компьютера. Канал 2 подключен к громкоговорителю компьютера и может быть использован для генерации различных звуков либо как генератор случайных чисел.

В ПК есть и другие устройства в ПК, позволяющие отсчитывать время:

Local APIC не имеет фиксированной известной частоты, она привязана к частоте ядра. Поэтому перед использованием программе необходимо её определить.

PMTIMER (Performance Monitoring Timer) – устройство, поддерживаемое всеми системами, реализующими стандарт ACPI, имеет частоту 3.579545 МГц, разрядность регистра-счётчика 24 или 32 бита.

HPET (High Precision Event Timer) — устройство, созданное как замена устаревшему PIT. HPET присутствует в PC с 2005 года, однако ОС его не используют из-за неудобного способа задания интервалов с помощью возрастающего счётчика вместо убывающего.

ACPI – усовершенствованный интерфейс управления конфигурацией и питанием, открытый промышленный стандарт, который определяет общий интерфейс для обнаружения аппаратного обеспечения, управления питанием и конфигурированием.

Тема 10 Базовая система ввода-вывода

- функции BIOS;
- процедура POST;
- звуковые сигналы и текстовые сообщения;
- главная загрузочная запись;
- виды разделов жесткого диска.

Литература [5, стр. 9-29], [6, стр. 36-54]

BIOS (англ. basic input/output system — «базовая система ввода-вывода») — реализованная в виде микропрограмм часть системного программного обеспечения, которая предназначается для обеспечения операционной системы API доступа к аппаратуре компьютера и подключенным к нему устройствам. В персональных IBM PC-совместимых компьютерах, использующих микроархитектуру x86, BIOS представляет собой набор записанного в ПЗУ микропрограмм.

Бóльшую часть BIOS материнской платы составляют микропрограммы инициализации контроллеров на материнской плате, а также подключённых к ней устройств, которые в свою очередь могут иметь управляющие контроллеры с собственными BIOS.

Сразу после включения питания или после аппаратного сброса во время начальной загрузки компьютера при помощи программ, записанных в BIOS, происходит самопроверка аппаратного обеспечения компьютера — POST. В ходе POST проверяется работоспособность контроллеров на материнской плате, задаются низкоуровневые параметры их работы, после чего ищется на доступных носителях загрузчик операционной системы и передаётся ему управление.

В случае сбоя во время прохождения POST BIOS может выдать информацию, позволяющую выявить причину сбоя. Кроме вывода сообщения на монитор, используется звуковой сигнал, воспроизводимый при помощи встроенного динамика.

Загрузчик операционной системы обеспечивает необходимые средства для диалога с пользователем компьютера (например, загрузчик позволяет выбрать операционную систему для загрузки); загружает ядро операционной системы в ОЗУ (загрузка ядра не обязательно происходит с жесткого диска, загрузчик может получать ядро по сети или загружаться через последовательные

интерфейсы); формирует параметры, передаваемые ядру операционной системы (например, ядру Linux передаются параметры, указывающие способ подключения корневой файловой системы); передаёт управление ядру операционной системы.

Для работы с жестким диском его для начала необходимо как-то разметить, чтобы операционная система могла понять в какие области диска можно записывать информацию. Поскольку жесткие диски имеют большой объем, их пространство обычно разбивают на несколько частей — разделов диска (partition). В таблице разделов хранится информация о типе раздела и его расположении на жёстком диске. Существует два типа разделов: первичный (их может быть до четырех, один должен быть помечен как активный) и расширенный, который может быть разделен на любое количество логических дисков.

С развитием компьютерных систем в BIOS продолжали использоваться устаревшие технологии: прежде всего «реальный режим» работы процессора x86. Для принципиальной замены BIOS рядом производителей вычислительных систем (Unified EFI Forum, UEFI) предложена и внедряется технология EFI.

Тема 11 UEFI

- недостатки BIOS;
- преимущества UEFI;
- GPT-разметка диска.

Литература [6, стр. 55-63]

Extensible Firmware Interface (EFI) (англ. Расширяемый интерфейс прошивки) — интерфейс между операционной системой и микропрограммами, управляющими низкоуровневыми функциями оборудования, его основное предназначение: корректно инициализировать оборудование при включении системы и передать управление загрузчику операционной системы. EFI предназначен для замены BIOS — интерфейса, который традиционно используется всеми IBM PC-совместимыми персональными компьютерами.

UEFI совершенно новый интерфейс, который полностью вскоре вытеснит BIOS, потому как BIOS имеет ряд серьезных недостатков:

- может загружаться с дисков объемом не более 2 Тб;

- BIOS работает в 16-битном режиме процессора и ему доступен всего 1 Мб памяти;
- большое количество производителей и их собственных версий BIOS;
- не имеет поддержки 64-ех разрядных систем.
- UEFI имеет следующие преимущества:
- понятная и удобная графическая оболочка с поддержкой манипулятора;
- поддержка винчестеров с таблицей разделов GPT и адресацией LBA;
- полноценная работа с жесткими дисками, объем которых превышает 2 Тбайта;
- увеличенная скорость загрузки системы;
- наличие менеджера загрузки, который предоставляет выбор операционной системы;
- простота обновления прошивки.

Использование UEFI вместо BIOS — это один из ключевых моментов в обеспечении быстрой загрузки операционной системы, а также защиты данных от выполнения вредоносного кода еще на этапе инициализации Windows. Модульная архитектура UEFI позволяет использовать свои приложения и загружать собственные драйвера при помощи UEFI Shell, а также управлять файловой системой, не загружая Windows.

Вместе с UEFI появился новый способ разметки жесткого диска GUID Partition Table (GPT). В структуре GPT используется только LBA-адресация. В отличие от MBR, структура GPT хранит на диске две своих копии, одну в начале диска, а другую в конце. Таким образом, в случае повреждения основной структуры, будет возможность восстановить ее из сохраненной копии.

Вся структура GPT на жестком диске состоит из следующих частей:

- Защитный MBR-сектор. Он оставлен для совместимости со старым программным обеспечением и предназначен для защиты GPT-структуры от случайных повреждений.
- Первичный GPT-заголовок. Этот заголовочный сектор содержит в себе данные о всех LBA-адресах, использующихся для разметки диска на разделы.
- Таблица разделов диска содержит информацию о всех возможных 128 разделах на диске (их адреса, атрибуты, права доступа), на каждую запись раздела выделяется по 128 байт.
- Содержимое разделов.

- Копия таблицы разделов хранит копию таблицы разделов в случаях проблем с основной, изменения в обе таблицы вносятся параллельно.
- Копия GPT-заголовка.

Apple MacBook используют GPT по умолчанию, так что невозможно установить MacOS X на систему MBR. Большинство операционных систем на ядре Linux совместимы с GPT. Для Windows загрузка из GPT возможна только с UEFI на компьютерах с Windows Vista, 7, 8, 10.

Тема 12 Подсистема прерываний

- классификация прерываний;
- приоритет прерывания;
- контроллер прерываний;
- организация обработки прерываний в ЭВМ.

Литература [12, стр. 112-122]

Прерывание – это прекращение выполнения текущей последовательности команд для обработки некоторого события специальной программой – обработчиком прерывания – с последующим возвратом к выполнению прерванной программы. Событие может быть вызвано особой ситуацией, сложившейся при выполнении программы, или сигналом от внешнего устройства. Прерывание используется для быстрой реакции процессора на такие особые ситуации, возникающие при выполнении программы и взаимодействии с внешними устройствами.

Механизм прерывания обеспечивается соответствующими аппаратно-программными средствами. Любая особая ситуация, вызывающая прерывание, сопровождается сигналом, называемым запросом прерывания. После появления сигнала запроса прерывания ЭВМ переходит к выполнению программы-обработчика прерывания. Обработчик выполняет те действия, которые необходимы в связи с возникшей особой ситуацией. Например, такой ситуацией может быть нажатие клавиши на клавиатуре. Тогда обработчик данного прерывания должен занести ASCII-код нажатой клавиши в буфер клавиатуры и вывести его на экран. По окончании работы обработчика управление передается прерванной программе.

Аппаратные прерывания используются для организации взаимодействия с внешними устройствами. Они бывают маскируемые, которые могут быть замаскированы программными средствами, и немаскируемые, запрос от которых нельзя замаскировать.

Программные прерывания вызываются следующими ситуациями:

- особый случай, возникший при выполнении команды и препятствующий нормальному продолжению программы (переполнение, нарушение защиты памяти, отсутствие нужной страницы в оперативной памяти и т.п.);
- наличие в программе специальной команды прерывания INT n, используемой обычно программистом при обращениях к специальным функциям операционной системы для ввода-вывода информации.

Каждому запросу прерывания присваивается свой номер, используемый для определения адреса обработчика прерывания. Эти номера соответствуют “важности” события в системе и определяют приоритет прерывания.

Сигналы прерываний приходят на специальную электронную схему ЦП, которая называется программируемым контроллером прерываний. При поступлении запроса прерывания контроллер прерываний выполняет следующую последовательность действий:

- определение наиболее приоритетного незамаскированного запроса на прерывание, если одновременно поступило несколько запросов;
- определение типа выбранного запроса;
- сохранение текущего состояния счетчика команд и регистра флагов;
- определение адреса обработчика прерывания по его номеру и передача управления обработчику;
- выполнение программы-обработчика прерывания;
- восстановление сохраненных значений счетчика команд и регистра флагов прерванной программы;
- продолжение выполнения прерванной программы.

Этапы 1-4 выполняются аппаратными средствами ЭВМ автоматически при появлении запроса прерывания. Этап 6 также выполняется аппаратно по команде возврата из обработчика прерывания.

Переход к соответствующему обработчику прерывания осуществляется посредством таблицы векторов прерываний. Эта таблица располагается в оперативной памяти и содержит соответствующие значения сегментного регистра команд и указателя команд для обработчиков прерываний.

Тема 13 Организация подсистемы прерываний в современных ПК

- расширенный контроллер прерываний APIC;
- технология MSI.

Литература [1, стр. 155-166], [14, стр. 171-173]

APIC (англ. Advanced Programmable Interrupt Controller) — улучшенный программируемый контроллер прерываний. APIC использовался в многоядерных/многопроцессорных системах, начиная с Intel Pentium. Преимущества расширенного контроллера прерываний:

- возможность реализации межпроцессорных прерываний – сигналов от одного процессора другому;
- поддержка до 256 входов IRQ;
- крайне быстрый доступ к регистрам текущего приоритета прерывания и подтверждения прерывания.

В многоядерных ЭВМ обычно у каждого процессорного ядра есть свой контроллер внутренних прерываний (LOCAL APIC) и один на все ядра контроллер внешних прерываний (I/O APIC). При этом каждое процессорное ядро после выполнения очередной своей команды "смотрит" на свой контроллер внутренних прерываний и на общий контроллер внешних прерываний. Обычно сигнал прерывания с такого общего внешнего контроллера направляется ядру, которое выполняет процесс с наименьшим приоритетом, эта информация предоставляется контроллеру операционной системой. При подаче сигнала прерывания некоторому ядру устанавливается наивысший приоритет выполняемого им процесса, так что следующий сигнал прерывания будет направлен уже другому ядру. Этот механизм позволяет равномерно распределять нагрузку по обработке внешних прерываний между процессорными ядрами. Необходимо также отметить, что одно ядро, выступая как внешнее устройство, может отправить сингал прерывания другому ядру.

Необходимость в новом контроллере, способном заменить программируемый контроллер прерываний (PIC), возникла с появлением следующих проблем:

- появление многоядерных систем, требующих распределения прерываний по ядрам;

- резкий рост числа подключенных устройств, превышающее количество свободных IRQ процессора;
- скорость передачи данных устройств, превышающая скорость работы PIC.

Расширенный контроллер прерываний впервые начал применяться на двухпроцессорных системных платах из-за более сложной обработки прерываний от различных устройств: не совсем очевидно, какой из процессоров должен реагировать на прерывание. Затем расширенный контроллер прерываний начал использоваться и на однопроцессорных системах: устройствам становится доступно большее число прерываний.

В настоящий момент наблюдается тенденция к отказу от IO APIC, как и проводников IRQ, и переходу на Message Signaled Interrupts

Message Signaled Interrupts (MSI, Прерывания, инициируемые сообщениями) — альтернативная форма прерываний, PCI версии 2.2 и более поздних и обязательная в PCI Express любых версий. Вместо присваивания фиксированного номера запроса на прерывание, устройству разрешается записывать сообщение по определённому адресу системной памяти, на деле отображенному на аппаратуру локального контроллера прерываний (local APIC) процессора.

Достоинства MSI:

- возможность полного отказа от проводников INT# от устройств, что упрощает материнскую плату.
- в многопроцессорных и многоядерных системах устройства, использующие несколько областей MSI, получают возможность самостоятельно выбирать процессор/ядро для обработки конкретного прерывания, причем делать это полностью на уровне аппаратуры без исполнения программного кода. Это позволяет оптимизировать работу путём размещения большей части структур драйвера устройства и связанного с ним программного обеспечения (сетевых протоколов) в кэше конкретного ядра.

MSI поддерживается в операционных системах Microsoft Windows Vista и более поздних, а также в ядре Linux , начиная с версии 2.6.8.

Тема 14 Подсистема ввода-вывода

- блочные и символьные устройства;
- состав и подключение устройств ввода-вывода;

- управление вводом-выводом;
- подсистема ввода-вывода.

Литература [13, стр. 59-69], [14, стр. 169-175]

Устройства ввода-вывода можно разделить на блочные и символьные. Блочными являются устройства, хранящие информацию в виде блоков фиксированного размера, причем у каждого блока есть адрес и каждый блок может быть прочитан независимо от остальных блоков. Символьные устройства принимают или передают поток символов без какой-либо блочной структуры (принтеры, сетевые карты, мыши и т.д.).

Устройства ввода-вывода, как правило, состоят из электромеханической и электронной части. Обычно их выполняют в форме отдельных модулей – собственно устройство и контроллер (адаптер). В ПК контроллер принимает форму платы, вставляемой в слот расширения, или входит в состав южного моста. Каждый контроллер взаимодействует с драйвером системным программным модулем, предназначенным для управления данным устройством.

Обмен данными между пользователями, приложениями и периферийными устройствами компьютера выполняет специальная подсистема ОС – подсистема ввода-вывода. Основными компонентами подсистемы ввода-вывода являются драйверы, управляющие внешними устройствами, и файловая система. В работе подсистемы ввода-вывода активно участвует контроллер прерываний.

Эволюция ввода-вывода может быть представлена следующими этапами:

1. Процессор непосредственно управляет периферийным устройством.
2. Устройство управляется контроллером. Процессор использует программируемый ввод-вывод без прерываний (переход к абстракции интерфейса ввода-вывода).
3. Использование контроллера прерываний. Ввод-вывод, управляемый прерываниями.
4. Использование модуля (канала) прямого доступа к памяти. Перемещение данных в память (из нее) без применения процессора.
5. Использование отдельного специализированного процессора ввода-вывода, управляемого центральным процессором.
6. Использование отдельного компьютера для управления устройствами ввода-вывода при минимальном вмешательстве центрального процессора.

Таким образом, постепенно процессор все больше освобождается от задач, связанных с вводом-выводом, что приводит к повышению общей производительности.

Для персональных компьютеров операции ввода-вывода могут выполняться тремя способами.

- с помощью программируемого ввода-вывода;
- ввод-вывод, управляемый прерываниями;
- прямой доступ к памяти.

Тема 15 Прямой доступ к памяти

- реализация технологии прямого доступа к памяти в ПК;
- состав контроллера ПДП;
- работа контроллера ПДП;
- режимы работы шины.

Литература [12, стр. 124-126], [14, стр. 174]

Прямой доступ к памяти (англ. direct memory access, DMA) — режим обмена данными между устройствами или между устройством и ОЗУ, в котором центральный процессор не участвует. Так как данные не пересылаются в ЦП и обратно, то скорость передачи увеличивается. Для организации такого режима имеется специальный DMA-контроллер, обслуживающий запросы по передаче данных от нескольких устройств ввода-вывода.

DMA-контроллер имеет доступ к системной шине независимо от центрального процессора, как показано на рисунке 8. Контроллер содержит несколько регистров, доступных центральному процессу для чтения и записи (регистр адреса памяти, счетчик байтов, управляющие регистры). Управляющие регистры задают порт ввода-вывода, который должен быть использован, направление переноса данных (чтение или запись в устройство ввода-вывода), единицу переноса (побайтно, пословно), а также число байтов, которые следует перенести за одну операцию.

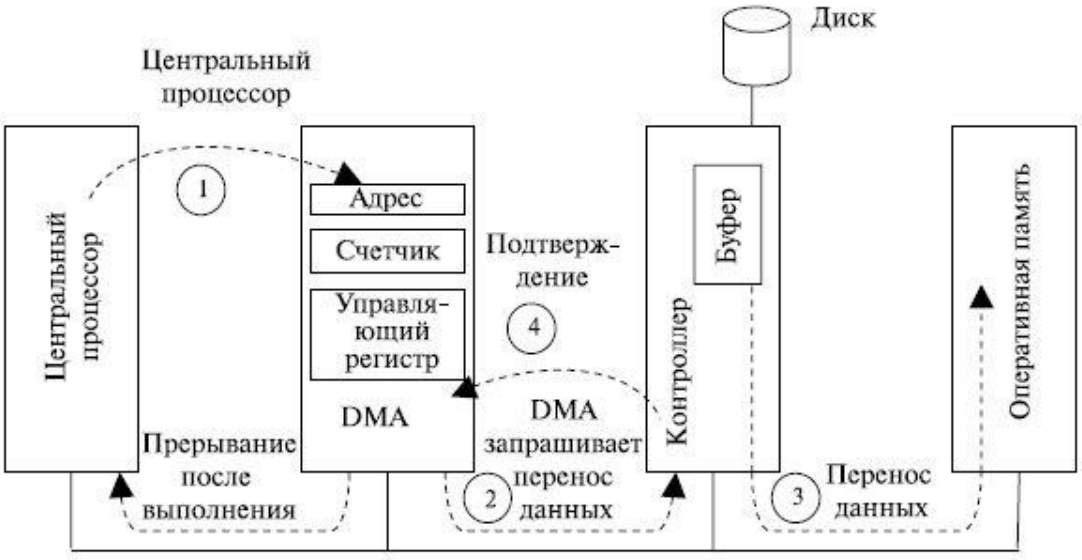

Рисунок 8 – Работа контроллера DMA

Необходимо обратить внимание на работу шины в этом процессе обмена данными. Шина может работать в двух режимах: пословном и поблочном. В первом случае контроллер DMA выставляет запрос на перенос одного слова и получает его. Если процессору также нужна эта шина, ему приходится подождать. Этот механизм называется захватом цикла потому, что контроллер устройства периодически забирает случайный цикл шины у центрального процессора, слегка тормозя его. В блочном режиме работы контроллер DMA занимает шину на серию пересылок (пакет). Этот режим более эффективен, однако при переносе большого блока центральный процессор и другие устройства могут быть заблокированы на существенный промежуток времени.

При большом количестве устройств ввода-вывода от подсистемы ввода-вывода требуется спланировать в реальном масштабе времени, в котором работают внешние устройства, запуск и приостановку большего количества разных драйверов, обеспечив при этом время реакции каждого драйвера на независимые события контролеров внешних устройств. С другой стороны, необходимо минимизировать загрузку процессора задачами ввода-вывода.

Решение этих задач достигается на основе многоуровневой приоритетной схемы обслуживания прерываний. Для обеспечения приемлемого уровня реакции все драйверы распределяются по нескольким приоритетным уровням в соответствии с требованиями по времени реакции и временем использования процессора. Для реализации приоритетной схемы задействуется общий диспетчер прерываний ОС.

Тема 16 Файловая система

- понятие файла;
- назначение файловой системы;
- файловая система FAT;
- файловая система NTFS.

Литература [13, стр. 118-150, 164-181]

Файл – это именованная часть диска, представляющая собой совокупность записей одинаковой структуры. Файловая система – это набор спецификаций и соответствующее им программное обеспечение, которые отвечают за создание, уничтожение, организацию, чтение, запись, модификацию и перемещение файловой информации, а также за управление доступом к файлам и за управлением ресурсами, которые используются файлами.

Для семейства ОС Windows в основном используются файловые системы FAT32, NTFS. Рассмотрим структуру этих файловых систем.

В файловой системе FAT дисковое пространство любого логического диска делится на две области: системную область и область данных.

Системная область создается и инициализируется при форматировании, а впоследствии обновляется при манипулировании файловой структурой. Системная область состоит из следующих компонентов:

- загрузочного сектора, содержащего загрузочную запись (boot record);
- зарезервированных секторов (их может и не быть);
- таблицы размещения файлов (FAT, File Allocation Table);
- корневого каталога (Root directory, ROOT).

Эти компоненты расположены на диске друг за другом.

Область данных содержит файлы и каталоги, подчиненные корневому. Область данных разбивают на так называемые кластеры. Кластер – это один или несколько смежных секторов области данных. Для создания и записи на диск нового файла операционная система отводит для него несколько свободных кластеров диска. Для каждого файла хранится список всех номеров кластеров, которые предоставлены данному файлу.

Основной недостаток FAT – медленная работа с файлами. При создании файла работает правило: выделяется первый свободный кластер. Это ведет к

фрагментации диска и сложным цепочкам файлов. Отсюда следует замедление работы с файлами.

Файловая система NTFS (New Technology File System) содержит ряд значительных усовершенствований и изменений, отличающих ее от других файловых систем.

Основные особенности NTFS:

- работа на дисках большого объема происходит намного эффективнее, чем в FAT;
- имеются средства для ограничения доступа к файлам и каталогам, разделы NTFS обеспечивают локальную безопасность как файлов, так и каталогов;
- введен механизм транзакций, при котором осуществляется журналирование файловых операций, существенное увеличение надежности;
- сняты многие ограничения на максимальное количество дисковых секторов;
- система NTFS также обладает встроенными средствами сжатия, которые можно применять к отдельным файлам, целым каталогам.

Раздел NTFS называется томом (volume). Максимально возможные размеры тома (и размеры файла) составляют 16 Эбайт (экзабайт 2^{64}).

Все дисковое пространство в NTFS делится на две неравные части. Первые 12 % диска отводятся под так называемую MFT-зону — пространство, которое может занимать, увеличиваясь в размере, главный служебный метафайл MFT. Остальные 88 % тома представляют собой обычное пространство для хранения файлов.

MFT (master file table, общая таблица файлов) – это каталог всех остальных файлов диска, в том числе и себя самого. Он предназначен для определения расположения файлов. MFT состоит из записей фиксированного размера. Размер записи MFT (минимум 1 Кб и максимум 4 Кб) определяется во время форматирования тома. Каждая запись соответствует какому-либо файлу.

Первые 16 записей носят служебный характер и недоступны операционной системе, они называются метафайлами, причем самый первый метафайл – сам MFT. В соответствующей записи MFT хранится вся информация о файле: имя, размер, атрибуты файла, положение на диске и т. д.

Тема 17 Подсистема памяти

- иерархия памяти;
- типовая структурная схема кэш-памяти;
- способы организации кэш-памяти;
- контроллер кэш-памяти;
- методы обновления строк.

Литература [1, стр. 277-287, стр. 473-492]

Запоминающие устройства компьютера разделяют на два типа: основную (главную, оперативную, физическую) и вторичную (внешнюю) память. Все запоминающие устройства ПК можно представить в виде иерархии слоев, как показано на рисунке 9.

Входящие в состав такой иерархии запоминающие устройства различаются емкостью, быстродействием и стоимостью (убывает время доступа, возрастает цена и увеличивается емкость). Более высокий уровень меньше по емкости, быстрее и имеет большую стоимость в пересчете на бит, чем более низкий уровень. По мере движения по иерархической структуре уменьшается соотношение «стоимость/бит», возрастает емкость, растет время доступа, уменьшается частота обращения к памяти со стороны центрального процессора.

Кэш-память отличается меньшим объемом, но более высоким быстродействием по сравнению с оперативной памятью. Она функционирует как буфер между процессором и оперативной памятью и содержит копии областей памяти, которые используются в данный момент или будут использоваться в ближайшее время.

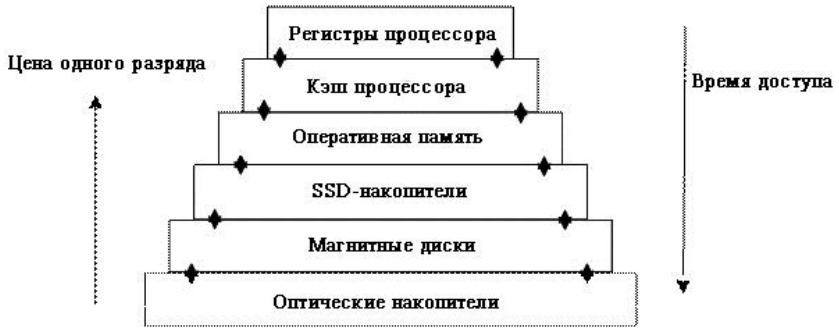

Рисунок 9 – Иерархия памяти

За единицу информации при обмене между основной памятью и кэш-памятью принята строка, причём под строкой понимается набор слов, выбираемый из оперативной памяти при одном к ней обращении. Хранимая в

оперативной памяти информация представляется, таким образом, совокупностью строк с последовательными адресами. В любой момент времени строки в кэш-памяти представляют собой копии строк из некоторого их набора в ОП, однако расположены они необязательно в такой же последовательности, как в ОП.

У кэш, как и у любого другого устройства, есть свой контроллер. Каждый раз, когда процессору требуется обратиться к ОЗУ по заданному адресу, он сначала обращается к контроллеру кэш-памяти, который проверяет, есть ли нужная строка в кэше. Если это так, то происходит результативное обращение к кэш-памяти, запрос удовлетворяется из кэша и запрос к памяти на шину не выставляется. Удачное обращение к кэшу, как правило, по времени занимает около двух тактов, а неудачное приводит к обращению к памяти с существенной потерей времени (сотни тактов). В этом случае данные считываются из памяти в указанный в команде регистр процессора, а их копия записывается в кэш.

Типовая структура кэш-памяти представлена на рисунке 10.

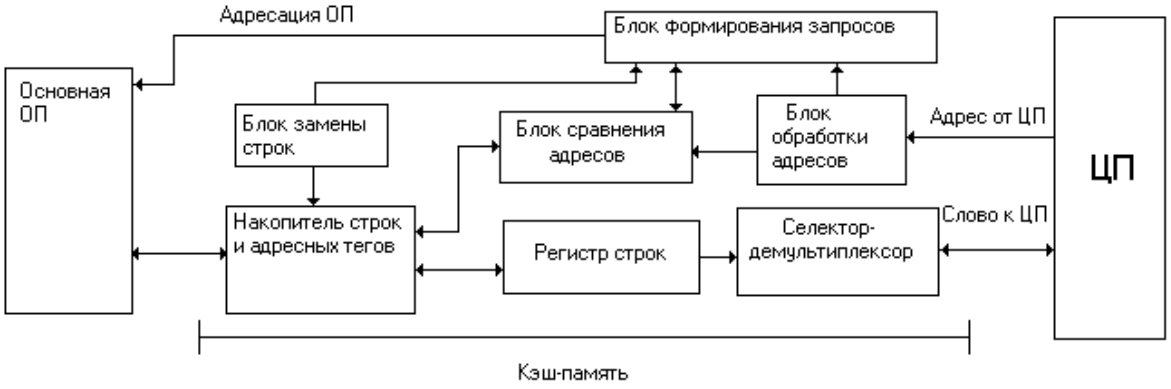

Рисунок 10 – Типовая структура кэш-памяти

Строки, составленные из информационных слов, и связанные с ними адресные теги хранятся в накопителе, который является основой кэш-памяти. Адрес требуемого слова, поступающий от центрального процессора (ЦП), вводится в блок обработки адресов, в котором реализуются принятые в данной кэш-памяти принципы использования адресов при организации их сравнения с адресными тегами. Само сравнение производится в блоке сравнения адресов (БСА). Назначение БСА состоит в выявлении попадания или промаха при обработке запросов от центрального процессора. Выбираемая из памяти строка вместе со своим адресным тегом помещается в накопитель и регистр строк, а затем искомое слово передается в центральный процессор. Для высвобождения

места в кэш-памяти с целью записи выбираемой из ОП строки одна из строк удаляется. Существует четыре способа размещения данных в кэш-памяти:

- прямое распределение,
- полностью ассоциативное,
- множественно ассоциативное.

Тема 18 Виртуальная память

- схемы организации оперативной памяти;
- виртуальная память;
- сегментная и страничная организации виртуальной памяти;
- таблица страниц;
- подкачка с жесткого диска;
- контроллер памяти.

Литература [1, стр. 288-299], [2, стр. 476-502]

Управление памятью поддерживается как аппаратно (в процессор включен специальный модуль управления памятью – контроллер памяти), так и программно (есть модуль ОС – менеджер памяти).

Первые ОС применяли очень простые методы управления памятью. Вначале каждый процесс пользователя должен был полностью поместиться в основной памяти и занимать непрерывную область, а система принимала к обслуживанию дополнительные пользовательские процессы до тех пор, пока все они одновременно помещались в основной памяти. Затем появился т.н. "простой свопинг" (система по-прежнему размещает каждый процесс в основной памяти целиком, но иногда сбрасывает образ некоторого процесса из основной памяти во внешнюю и заменяет его образом другого процесса). Концепция виртуальной памяти была реализована в 1959 г. на компьютере "Атлас", разработанном в Манчестерском технологическом университете. Она предлагает механизм разделения небольшой физической памяти между различными задачами.

Суть концепции виртуальной памяти заключается в следующем. Информация, с которой работает активный процесс, должна располагаться в оперативной памяти. В схемах виртуальной памяти у процесса создается иллюзия того, что вся необходимая ему информация имеется в ОЗУ. Для этого, во-первых, занимаемая процессом память разбивается на несколько частей. Во-вторых, логический адрес, к которому обращается процесс, динамически

преобразуется в физический адрес (этот процесс называется трансляцией адреса). И, наконец, когда часть программы, к которой обращается процесс, не находится в физической памяти, организовывается ее подкачка с жесткого диска, то есть организуется механизм страничного обмена: не используемые в течение определенного промежутка времени страницы в ОЗУ, с которыми работает приложение, выгружаются из ОЗУ на жесткий диск и заменяются другими страницами, которые нужны для выполнения текущего приложения. Таким образом, экономится память компьютера, и это позволяет выполняться многим процессам одновременно. Для хранения страниц виртуальной памяти на диске создается т.н. swap file. От его размера зависит объём физической памяти, доступный приложениям, и, соответственно, производительность их работы.

Системы виртуальной памяти можно разделить на несколько типов: системы со страничным, сегментным и со странично-сегментным распределением памяти. Рассмотрим эти типы организации виртуальной памяти.

В случае страничной организации памяти как логическое (виртуальное) адресное пространство, так и физическое делятся на блоки или страницы одинакового размера. Каждое слово в виртуальной памяти пользователя определяется виртуальным адресом, состоящим из двух частей: старшие разряды адреса рассматриваются как номер страницы, а младшие как номер слова (или байта) внутри страницы. То есть логический адрес в страничной системе – упорядоченная пара (p,d), где p – номер страницы в виртуальной памяти, а d – смещение в рамках страницы p, на которой размещается адресуемый элемент. Для указания соответствия между виртуальными страницами и страницами основной памяти операционная система должна сформировать таблицу страниц для каждой программы и разместить ее в основной памяти машины. Каждый элемент таблицы страниц содержит номер физической страницы основной памяти и специальный индикатор. Единичное состояние этого индикатора свидетельствует о наличии этой страницы в основной памяти. Нулевое состояние индикатора означает отсутствие страницы в оперативной памяти.

Аппаратная организация памяти в виде линейного набора ячеек не соответствует представлениям программиста о том, как организовано хранение программ и данных. Большинство программ представляет собой набор модулей, созданных независимо друг от друга. Чаще модули помещаются в разные области памяти и используются по-разному. Схема управления памятью,

поддерживающая этот взгляд пользователя на то, как хранятся программы и данные, называется сегментацией. Сегмент – область памяти определенного назначения, внутри которой поддерживается линейная адресация. Сегменты содержат процедуры, массивы, стек, но обычно не содержат информацию смешанного типа.

Таким образом, существуют еще две другие схемы организации управления памятью: сегментная и сегментно-страничная. Сегменты, в отличие от страниц, могут иметь переменный размер. При сегментной организации виртуальный адрес является двумерным как для программиста, так и для операционной системы, и состоит из двух полей – номера сегмента и смещения внутри сегмента.

Хранить в памяти сегменты большого размера целиком так же неудобно, как и хранить процесс непрерывным блоком. Напрашивается идея разбиения сегментов на страницы. При сегментно-страничной организации памяти происходит двухуровневая трансляция виртуального адреса в физический. В этом случае логический адрес состоит из трех полей: номера сегмента логической памяти, номера страницы внутри сегмента и смещения внутри страницы. Соответственно, используются две таблицы отображения – таблица сегментов, связывающая номер сегмента с таблицей страниц, и отдельная таблица страниц для каждого сегмента. В таблице страниц одержит информацию об атрибутах страницы. Это биты присутствия и защиты (например, 0 – read/write, 1– read only...). Также могут быть указаны: бит модификации, который устанавливается, если содержимое страницы модифицировано, и позволяет контролировать необходимость перезаписи страницы на диск; бит, разрешающий кэширование, и другие управляющие биты.

Любая из рассмотренных схем управления памятью пригодна для организации виртуальной памяти. Чаще всего используется сегментно-страничная модель, которая является синтезом страничной модели и идеи сегментации. Причем для тех архитектур, в которых сегменты не поддерживаются аппаратно (аппаратная поддержка сегментов распространена на процессорах Intel), их реализация – задача архитектурно-независимого компонента менеджера памяти. Сегментная организация в чистом виде встречается редко.

Тема 19 Накопитель на жестких магнитных дисках

- состав и функционирование накопителя;
- основы магнитной записи;
- адаптер НЖМД.

Литература [4, стр. 150-157], [7, стр. 187-215]

Накопитель на жестком магнитном диске (НЖМД) – запоминающее устройство произвольного доступа, основанное на принципе магнитной записи. Жёсткий диск состоит из гермозоны и блока электроники.

Гермозона включает в себя корпус из прочного сплава, диски (пластины) с магнитным покрытием, а также блок головок с устройством позиционирования и электропривод шпинделя. Большинство устройств содержит две-три пластины. Диски, как правило, изготовлены из металлического сплава и покрыты тонким слоем вещества – ферромагнетика, – способного сохранять остаточную намагниченность после воздействия внешнего магнитного поля. Этот слой называется рабочим или магнитным, и именно в нем сохраняется записанная информация. Блок головок чтения/записи – пакет кронштейнов (рычагов) из сплавов на основе алюминия, совмещающих в себе малый вес и высокую жёсткость (обычно по паре на каждый диск). Одним концом они закреплены на оси рядом с краем диска.

В каждом накопителе есть плата управления, на которой находится адаптер накопителя. Типичный адаптер НЖМД выполняет следующие функции по командам ЦП: поддерживает требуемый формат данных, осуществляет поиск и проверку требуемых цилиндров, производит переключение головок, обнаруживает и корректирует ошибки в считанных данных, организует последовательность считываемых секторов в соответствии с коэффициентом чередования, генерирует прерывание. На плате управления можно выделить управляющий блок, интерфейсный блок и блок цифровой обработки сигнала, а также постоянное запоминающее устройство и буферную память.

Принцип работы ЖД основан на понятии магнитного домена. Домен – это макроскопическая область в магнитном кристалле, в которой ориентация вектора однородной намагниченности определенным образом повернута или сдвинута относительно направлений соответствующего вектора в соседних доменах.

Таким образом, материал магнитного покрытия можно представить множеством хаотически расположенных магнитных доменов, ориентация которых изменяется под действием магнитного поля. Магнитная головка чтения позволяет определить моменты времени, когда при движении носителя под ней оказываются границы между участками с противоположными состояниями намагниченности.

Тема 20 Организация информации на магнитном диске

- [c h s] адресация;
- LBA-адресация;
- интерфейсы НЖМД.

Литература [7, стр. 191-195]

Существует два механизма доступ к данным на жестком магнитном диске.

[c h s] – [цилиндр головка сектор] – механизм доступа к данным на накопителе с учетом его геометрии. Дорожка – это одно "кольцо" данных на одной стороне диска. Блок дорожек одинакового радиуса образуют цилиндр. Дорожки на диске разбивают на нумерованные отрезки, называемые секторами. Каждый сектор на диске обычно занимает 571 байт, из которых под данные отводится только 512 байт, остальное – служебная информация, благодаря которой контроллер идентифицирует начало и конец сектора.

LBA (Logical block addressing) – механизм адресации и доступа к блоку данных на жёстком диске, при котором системному контроллеру нет необходимости учитывать геометрию самого диска (количество цилиндров, сторон (головок), секторов на дорожке). Контроллеры современных дисков в качестве основного режима трансляции адреса используют LBA. Суть LBA состоит в том, что каждый блок, адресуемый на жёстком диске, имеет свой номер, целое число, начиная с нуля (первый блок LBA=0, второй LBA=1), при этом LBA 0 = Цилиндр 0/Головка 0/Сектор 1.

Преимущество LBA – отсутствие ограничения размера диска. На сегодня есть возможность адресовать на приводе (248) 281 474 976 710 656 блоков (то есть при блоке в 512 байт, 128 ПиБ (пебибайт – 250)).

Физический порядок секторов на диске не изменяется: за сектором n следует сектор n+1 при любой адресации, CHS и LBA. Разные способы адресации означают лишь различные алгоритмы трансляции.

Интерфейсы подключения накопителей: PATA, SCSI, SATA, SAS, USB, Thunderbolt, FireWire.

Тема 21 Видеосистема

- состав и параметры видеосистемы;
- назначение графического адаптера;
- назначение графического контроллера;
- режимы организации видеопамяти;
- характеристики видеоадаптера.

Литература [4, стр. 136-147], [7, стр. 289-310]

Видеосистема персонального компьютера предназначена для формирования изображений, наблюдаемых на экране монитора. Видеосистема компьютера – совокупность трех компонент: видеоадаптера, формирующего изображение, монитора, выводящего изображение, и драйверов видеосистемы, представляющих собой набор программ, поставляемых вместе с адаптером.

Адаптер состоит из следующих компонентов: графический процессор, графический контроллер, видеопамять, видео-ПЗУ, системы охлаждения. Графический процессор (англ. GPU, Graphics processing unit – графическое процессорное устройство) характеризует быстродействие адаптера и его функциональные возможности, занимается расчётами выводимого изображения, производит расчёты для обработки команд трёхмерной графики, освобождая от этой обязанности центральный процессор. Графический контроллер преобразует информацию, содержащуюся в видеопамяти в уровни напряжения, которые будут подаваться на монитор.

Видеопамять является оперативным запоминающим устройством. Она временно хранит изображение, которое рассчитывает графический процессор, а также некоторые промежуточные результаты этих вычислений. Кроме видеопамяти графические адаптеры в процессе работы вполне могут использовать и часть оперативной памяти компьютера. Видеопамять может быть организована двумя способами: в текстовом режиме (при загрузке операционной

системы или в режиме виртуальной машины), где видеопамять разбивается на знакоместа и хранит информацию об ASC|II-коде символа каждого знакоместа и его атрибутах, и в графическом (основной режим работы ПК), где изображение представляет собой матрицу пикселей, а видеопамять хранит значение RGB-цветов каждого пиксела.

Видео-ПЗУ – постоянное запоминающее устройство, в которое записаны видео-BIOS, экранные шрифты, служебные таблицы и т.п.

Система охлаждения представляет собой набор средств для отвода тепла от нагревающихся в процессе работы компонентов.

К характеристикам видеоадаптера относятся тип интерфейса подключения монитора (DVI, HDMI, Display Port), интерфейс подключения видеокарты (PCI-Exress), объем тип, разрядность видеопамяти, а также разрядность и быстродействие чипсета видеоадаптера.

Тема 22 Мониторы

- понятие люминофора;
- конструкция и принцип работы ЭЛТ-монитора;
- достоинства и недостатки ЭЛТ-монитора;
- поляризация света;
- принцип работы ЖК-панели;
- характеристики ЖК-мониторов.

Литература [4, стр. 180-192], [7, стр. 310-312]

CRT (Cathode Ray Tube) или ЭЛТ-монитор (монитор с электронно-лучевой трубкой) имеет стеклянную трубку, внутри которой находится вакуум. С фронтальной стороны внутренняя часть стекла трубки покрыта люминофором. Люминофор – это вещество, которое испускает свет при бомбардировке его заряженными частицами. Для цветных мониторов используются три типа разноцветных частиц, чьи цвета соответствуют основным цветам RGB и при попадании на них заряженных частиц в зависимости от своего химического состава излучают красный, зеленый или синий свет. Эти светящиеся точки люминофора формируют изображение, которое видно на мониторе.

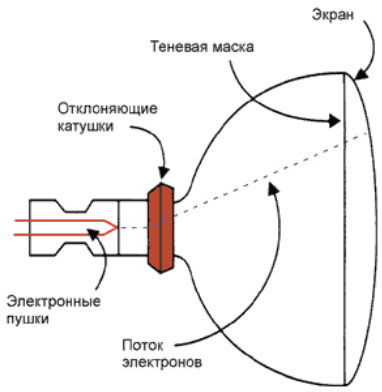

Рисунок 11 – Конструкция ЭЛТ-монитора

Для создания изображения в CRT мониторе используется электронная пушка, которая испускает поток электронов сквозь металлическую маску/решетку на внутреннюю поверхность стеклянного экрана монитора. В цветном CRT-мониторе используется три электронных пушки, расположенные в основании горловины. Каждая из трех пушек соответствует одному из основных цветов и посылает пучок электронов на различные частицы люминофора, чье свечение основными цветами с различной интенсивностью комбинируется. Для более точного позиционирования лучей на точке используется специальная маска – устройство, служащее для того, чтобы электронный луч каждой пушки попадал на люминофор только одного какого-либо цвета и не возбуждал другие точки.

Основные достоинства CRT монитора: качественная передача цветов без искажений, большой угол обзора, высокая скорость отклика, глубокий черный цвет, повышенная контрастность и яркость. Основные недостатки CRT монитора: существенные физические габариты, проблема с отображением геометрических фигур и их пропорций, сильное излучение, повышенное энергопотребление.

Экраны LCD-мониторов (*англ.* Liquid Crystal Display – жидкокристаллические мониторы) сделаны из вещества (цианофенил), которое находится в жидком состоянии, но при этом обладает некоторыми свойствами, присущими кристаллическим телам. Фактически это жидкости, обладающие анизотропией свойств (в частности оптических), связанных с упорядоченностью в ориентации молекул.

Работа ЖК-дисплея основана на явлении поляризации светового потока. Известно, что так называемые кристаллы-поляроиды (жидкие кристаллы)

способны пропускать только ту составляющую света, вектор электромагнитной индукции которой лежит в плоскости, параллельной оптической плоскости поляроида. Для оставшейся части светового потока поляроид будет непрозрачным. Таким образом поляроид как бы фильтрует свет. Данный эффект называется поляризацией света. Таким образом, принцип работы любого жидкокристаллического экрана основан на свойстве жидких кристаллов изменять (поворачивать) плоскость поляризации проходящего через них света пропорционально приложенному к ним напряжению. Если на пути поляризованного света, прошедшего через жидкие кристаллы, поставить поляризационный светофильтр (поляризатор), то, изменяя величину приложенного к жидким кристаллам напряжения, можно управлять количеством света, пропускаемого поляризационным светофильтром (смотри рисунок 12).

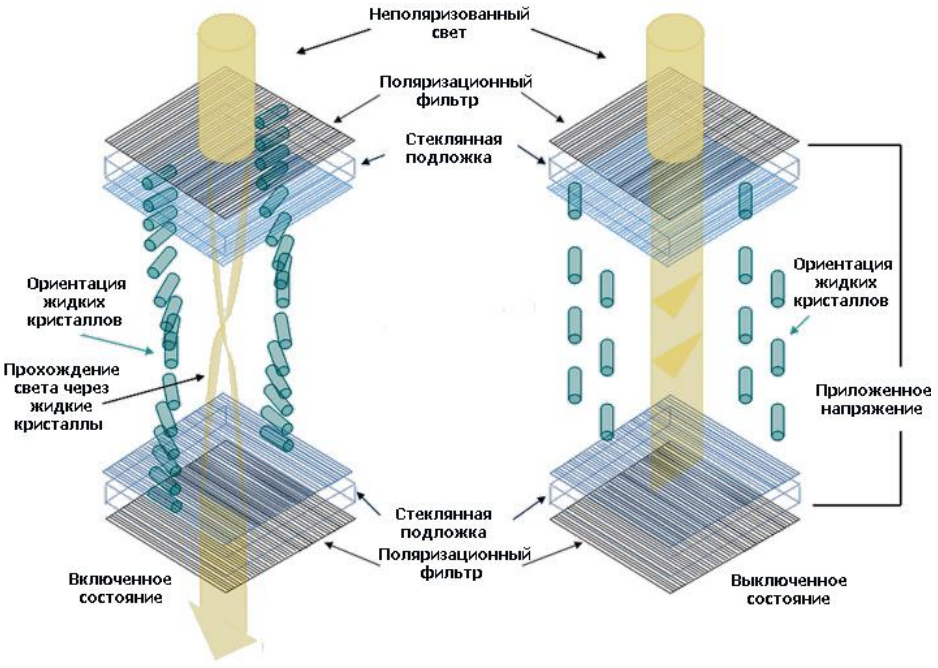

Рисунок 13 – Принцип работы ЖК-панели

Матрица ЖК-монитора – это массив, состоящий из множества мельчайших сегментов – пикселей, каждый из которых состоит из трех основных RGB-цветов – субпикселей. Каждым из этих субпикселей можно управлять в отдельности, благодаря чему и возникает определенная картинка.

Главные характеристики ЖК-мониторов: разрешение, размер точки, соотношение сторон экрана, яркость, диагональ, контрастность, время отклика, угол обзора, тип матрицы при изготовлении монитора – TN, IPS, MVA.

Тема 23 Устройства ввода

- классификация устройств ввода;
- кодирование текстовой информации;
- кодирование графической информации;
- принцип работы клавиатуры;
- контроллеры клавиатуры.

Литература [4, стр. 192-200]

Устройства ввода – периферийное оборудование, предназначенное для ввода данных и сигналов в компьютер или в другое электронное устройство во время его работы. Устройства ввода по способу ввода информации можно подразделить на два основных класса: с клавиатурным вводом, при котором осуществляется ручной ввод с клавиатуры (клавиатура), и с прямым вводом, при котором данные вводятся специальными устройствами (устройства ввода графической/звуковой информации; игровые устройства ввода; указательные (координатные) устройства; непрерывные устройства ввода; устройства, основанные на компьютерном зрении).

Современный компьютер может обрабатывать числовую, текстовую, графическую, звуковую и видеоинформацию. Все это возможно, потому как все виды информации представлены в двоичном коде, потому что удобно технически реализовать информацию в виде последовательности электрических импульсов: импульс отсутствует (0 – напряжение 0.4В-0.6В), импульс есть (1 - напряжение 2.4В-2.7В).

Текстовая информация состоит из символов: букв, цифр, знаков препинания, символов псевдографики. Для того, чтобы закодировать один символ используют количество информации равное 1 байту. Таким образом, можно закодировать $2^8 = 256$ различных символов. Для кодировки русских букв используют пять различных кодовых таблиц (КОИ - 8, Windows-1251, Mac, ISO, Unicode). Все эти кодировки продолжают кодовую таблицу стандарта ASCII, кодирующую первых 128 символов (от 0 до 127).

В процессе кодирования графического изображения производится его пространственная дискретизация, и изображение разбивается на отдельные маленькие фрагменты (точки), и каждому фрагменту присваивается значение его цвета. Такое представление каждой точки изображения в виде цветового триплета RGB и хранится в файлах графических изображений.

Клавиатура – основное устройства ручного ввода данных в ЭВМ, представляющее собой совокупность датчиков, воспринимающих давление (прикосновение) на клавиши и замыкающих тем или иным способом определенную электрическую цепь.

Для управления работой клавиатуры используют контроллеры. Это позволяет упростить аппаратуру, увеличить ее возможности, повысить надежность. Вертикальные и горизонтальные шины матрицы контактов линии передачи сформированного скан-кода подключаются к контроллеру, находящемуся непосредственно в клавиатуре. Контроллер работает под управлением программы, хранимой в ПЗУ. Основное назначение этого контроллера – сформировать и передать скан-код системному контроллеру клавиатуры, находящемуся на материнской плате в составе южного моста. Когда скан-код поступает в системный контроллер клавиатуры, то инициализируется аппаратное прерывание IRQ1 и выполняется процедура обработки прерывания. Данное прерывание обслуживается специальной программой-обработчиком, входящей в BIOS. Процедура обработки прерывания включает в себя преобразование полученного скан-кода в ASCII, передача его в буфер клавиатуры и слежение за состоянием служебных клавиш.

Тема 24 Устройства вывода

- классификация устройств вывода;
- принцип работы матричного принтера;
- принцип работы струйного принтера;
- принцип работы лазерного принтера.

Литература [7, стр. 233-277]

Устройства вывода – периферийные устройства, преобразующие информацию в форму, удобную для восприятия человеком. Выделяют устройства вывода графической информации (принтер, плоттер, проектор), устройства ввода звуковой информации (колонки, наушники) и прочие.

Принтер – это внешнее периферийное устройство компьютера, предназначенное для вывода текстовой или графической информации, хранящейся в компьютере, на твёрдый физический носитель, обычно бумагу или

полимерную плёнку. По своему принципу действия принтеры делятся на матричные, струйные и лазерные.

Матричные принтеры – это принтеры ударного действия. Печатающая головка матричного принтера состоит из матрицы иголок (обычно 9 или 24), которые под воздействием магнитного поля "выталкиваются" из головки и ударяют по бумаге (через красящую ленту). Перемещаясь, печатающая головка оставляет на бумаге строку символов. Недостатки матричных принтеров: низкая скорость и невысокое качество печати, сильный шум. Достоинство: невысокая стоимость.

Принцип действия струйных принтеров похож на матричные принтеры тем, что изображение на носителе формируется из точек. Но вместо головок с иголками в струйных принтерах используется матрица дюз, печатающая жидкими красителями. Существуют два способа технической реализации способа распыления красителя: пьезоэлектрический и термический. Достоинства: низкий шум, достаточная скорость печати, приемлемая стоимость.

Принцип работы лазерного принтера заключается в следующем. По поверхности фотобарабана коротроном равномерно распределяется статический заряд, после этого светодиодным лазером (в светодиодных принтерах — светодиодной линейкой) в нужных местах этот заряд снимается засветкой: тем самым на поверхность фотобарабана помещается скрытое изображение. Далее на фотобарабан наносится тонер. Тонер притягивается к разряженным участкам поверхности фотобарабана, сохранившей скрытое изображение. После этого под фотобарабаном протягивается бумага, и тонер переносится на бумагу. После этого бумага проходит через блок термозакрепления, где тонер под температурой фиксируется в структуре бумаги. Далее с бумаги снимается электростатика и она поступает на выход устройства. Фотобарабан очищается от остатков тонера в узле очистки и цикл печати возобновляется. Достоинства: высокая скорость и качество печати, невысокая стоимость отпечатка. Недостаток: неэкологичность.

2. ПРАКТИЧЕСКИЙ РАЗДЕЛ

ЛАБОРАТОРНАЯ РАБОТА №1-2
«СОЗДАНИЕ ПРИЛОЖЕНИЙ ВЗАИМОДЕЙСТВИЯ С УСТРОЙСТВАМИ ВВОДА-ВЫВОДА. ВЫВОД ИНФОРМАЦИИ ОБ УСТАНОВЛЕННОМ ОБОРУДОВАНИИ»

Цель работы: изучить особенности вызова подгружаемых модулей ядра в ОС GNU/Linux, получить практические навыки создания простейших модулей ядра на примере обращения к системной области памяти CMOS для определения конфигурации установленного оборудования.

1. Теоретические сведения

1.1 Режим функционирования ядра ОС

Ядро Linux - это центральная часть большой и сложной операционной системы. При этом, несмотря на колоссальные размеры, оно имеет четкую структурную организацию в виде подсистем и уровней. Операционную систему можно условно разделить на два уровня. На верхнем уровне находится пользовательское пространство (пространство приложений пользователя). Под пользовательским пространством располагается пространство ядра Linux.

Микропроцессоры Intel начиная с модели 80386 поддерживают аппаратную защиту памяти, благодаря которой пользовательские приложения изолированы друг от друга. Эта защита реализована в виде четырех режимов работы процессора, которые часто называют «кольцами защиты». Большинство современных ОС (включая Linux) используют два из них:

наивысший уровень (нулевое кольцо, известное так же под названием привилегированный режим, или пространство ядра) – в нем работает ядро операционной системы;

низший уровень (пользовательский режим, пространство приложений пользователя) – в нем работают прикладные программы.

Таким образом, ядро и пользовательские приложения располагаются в разных защищенных адресных пространствах.

Каждый процесс в пространстве пользователя имеет свое собственное (виртуальное) адресное пространство, с адресацией, начинающейся с нуля.

Конечно на самом деле все процессы занимают разные участки оперативной памяти. При обращении к памяти виртуальный адрес автоматически транслируется в физический адрес, т. е. отображается на ту физическую область памяти, где на самом деле лежат данные процесса.

Ядро ОС работает по-другому: благодаря привилегированному режиму ему доступна вся физическая память, и в т.ч. участки памяти, выделенные прикладным процессам.

Поскольку прикладной процесс работает с виртуальными адресами, он не имеет доступа в пространство ядра, не может обращаться к памяти ядра и не может вызывать функции в ядре. Точно так же прикладной процесс не имеет доступа к некоторым привилегированным инструкциям процессора и специализированным регистрам (например, обеспечивающим защиту памяти). Наконец, прикладному процессу запрещено обращаться к адресному пространству портов ввода-вывода. Напомним, что порты ввода-вывода – это специальное адресное пространство, на которое отображаются встроенные ячейки памяти (регистры) микроконтроллеров всех устройств компьютера: клавиатуры, жестких дисков и т. д. Через порты ввода-вывода идет весь обмен данными с периферийными устройствами ЭВМ.

Поэтому если программисту требуется получить доступ к защищенным элементам компьютера – таким, как порты ввода-вывода – ему приходится писать не обычную прикладную программу, а модуль ядра, запускаемый Linux в привилегированном режиме. Поскольку модули ядра работают в общем адресном пространстве и имеют полный набор привилегий, при их написании нужно соблюдать осторожность: любое некорректное действие модуля может привести к краху системы.

В этом практикуме мы будем использовать механизм модулей ядра для доступа к различным аппаратным подсистемам компьютера. Защищенный режим процессора несколько усложняет работу с оборудованием; однако всё современное оборудование и всем современные операционные системы рассчитаны на работу в этом режиме. Альтернатива — попытка использования устаревших однозадачных ОС на современном оборудовании или эмуляция устаревших компьютеров в виртуальных машинах — сильно оторваны от реальных задач и потому не представляют практического интереса.

1.2 Разработка модуля ядра

Модуль — это программный код, который может быть загружен или выгружен ядром по мере необходимости. Модули расширяют функциональные возможности ядра без необходимости перезагрузки системы. Одна из разновидностей модулей ядра – драйверы устройств.

Один и тот же код можно скомпилировать и в виде модуля ядра, и в виде постоянно присутствующей в ядре части. Механизм модулей удобен, потому что иначе для добавления в ядро новых возможностей пришлось бы изменять код ядра, заново компилировать все ядро, а потом перезагружать систему.

На практике при конфигурировании ядра перед его компиляцией, можно определить, какие части ядра должны быть скомпилированы в объектном файле ядра, какие – в виде отдельных объектных файлов (т.е. модулей), а какие не должны компилироваться вовсе.

Код простейшего модуля ядра

Рассмотрим исходный текст простого модуля ядра (назовем его файл hello1.c).

```
#include <linux/module.h> /*Необходим для любого модуля ядра */
#include <linux/kernel.h> // Здесь находится определение KERN_INFO
int init_module(void)
{
printk(KERN_INFO "Hello world.\n");
/*
* Если вернуть ненулевое значение это будет воспринято как признак
* ошибки,  возникшей в процессе работы  init_module; модуль не
будет * загружен
*/
    return 0;
}
void cleanup_module(void)
{
    printk(KERN_ALERT "Goodbye world.\n");
}
```

Как можно догадаться, модуль выводит на экран сообщение «Hello world» при своей загрузке в память и сообщение «Goodbye world» при выгрузке из памяти.

Любой модуль ядра должен иметь хотя бы две функции: функцию init_module(), которая всегда вызывается во время загрузки модуля в память, и функцию завершения работы модуля — cleanup_module().

Обычно функция init_module() выполняет регистрацию обработчика какого-либо события или замещает какую-либо функцию в ядре своим кодом. Функция cleanup_module() является полной противоположностью, она отменяет изменения, сделанные функцией init_module().

Любой модуль ядра должен подключать заголовочный файл linux/module.h. В нашем примере мы подключали еще один файл — linux/kernel.h, но лишь для того, чтобы получить доступ к макроопределениям KERN_ALERT и KERN_INFO, которые необходимы для функции вывода printk.

Функция printk является аналогом стандартной функции printf и предназначена для вывода ядром различных системных сообщений на специальную виртуальную «консоль ядра». В процессе работы ядро Linux постоянно выводит различную информацию (информационные сообщения, сообщения об ошибках и т. д.). Поскольку операционная система предназначена для одновременной работы с большим количеством консолей / терминалов (как виртуальных, так и физических), то чтобы не затеряться среди многочисленных терминалов, сообщения ядра выводятся в специальный буфер – «консоль ядра». Далее эти сообщения сохраняются в различных системных журналах в подкаталоге /var/log (при этом сообщения разного типа попадают в разные журналы). Кроме того, последние сообщения ядра можно просмотреть, выполнив команду **dmesg**.

Именно в консоль ядра попадут сообщения, выведенные модулем при загрузке и выгрузке.

Загрузку модуля в память можно выполнить вручную на работающей системе с помощью системной утилиты **insmod** (задав ей имя модуля в качестве параметра командной строки). Выгрузку модуля из памяти можно выполнить с помощью аналогичной утилиты **rmmod**.

Сборка модуля ядра

Чтобы модуль был работоспособен, его необходимо скомпилировать. Для компиляции модуля ядра необходимо передать компилятору **gcc** ряд дополнительных опций. Для углубленного изучения предмета можно найти сведения по сборке модулей, которые не являются частью официального ядра, в

каталоге с исходными кодами Linux (который в разных системах находится либо в /usr/src/linux, либо в /usr/local/src/linux), в файле linux/Documentation/kbuild/modules.txt. Дополнительную информацию по оформлению Makefile'ов модулей можно найти в linux/Documentation/kbuild/makefiles.txt.

Чтобы собрать модуль hello.c, рассмотренный выше, можно использовать Makefile из одной строки, добавляющей сборку нашего модуля hello-1 в цель obj-m (специальную цель, предописанную в системе сборки ядра для компиляции объектного кода модулей):

```
obj-m += hello1.o
```

Для того, чтобы запустить процесс сборки модуля, можно воспользоваться командой

```
make -C /usr/src/linux-headers-`uname -r` SUBDIRS=$PWD modules
```

ПРИМЕЧАНИЕ 1: ключом «**-C** *dir»* мы изменяем для make рабочий каталог на каталог *dir;* кроме того мы подключаем к рабочему каталогу make текущий каталог (возвращаемый системной переменной $PWD) в качестве «подкаталога», также подлежащего обработке. Этот трюк позволяет не копировать исходный код нашего модуля в подкаталог с исходными кодами всего ядра.

ПРИМЕЧАНИЕ 2: обратите внимание, что в командной строке для компиляции модуля использована не обычая, а т. н. «обратная» кавычка. Клавиша, которая ее воспроизводит, на большинстве клавиатур расположена над клавишей табуляции.

ПРИМЕЧАНИЕ 3: напомним, что фрагмент командной строки, взятый в обратные кавычки, сначала выполняется в фоновом режиме, а потом то, что он вывел «на экран», подставляется в оригинальную командную строку. В нашем случае команда `uname -r` выводит версию сборки ядра Linux: например, если версия «3.2.0-33generic», то результатом подстановки «/usr/src/linux-headers-`uname -r`» будет «/usr/src/linuxheaders-3.2.0-33-generic».

При компиляции модуля на экран должно быть выведено нечто подобное:

```
[root@pcsenonsrv test_module]# make -C /usr/src/linux-`uname -r` SUBDIRS=$PWD modules
make: Entering directory `/usr/src/linux-2.6.x
     CC [M]  /root/test_module/hello1.o
```

```
      Building modules, stage 2.
      MODPOST
      CC         /root/test_module/hello1.mod.o
      LD [M]     /root/test_module/hello1.ko
make: Leaving directory `/usr/src/linux-2.6.x
```

В результате компиляции будет создан бинарный файл модуля с расширением «.ko».

Для загрузки модуля выполняется команда **sudo insmod ./hello1.ko** (появляющиеся сообщения о "загрязнении" ядра можно игнорировать на данном этапе разработки модуля).

ПРИМЕЧАНИЕ: обычно модули ядра устанавливаются в системный каталог /lib/modules/, и тогда модуль можно загружать в память только по его имени, без расширения и пути. Мы указываем точное расположение файла, чтобы обойти сложности с правами на запись в системный каталог.

Любой модуль ядра, загруженный в данный момент, отображается в подкаталоге /proc/modules. Заглянув туда, можно убедиться, что модуль стал частью ядра, по наличию там файла с именем модуля. Подробнее о назначении каталога /proc и его подкаталогов будет рассказано ниже.

Выгрузить модуль можно командой **sudo rmmod hello1.**

Сообщения, которые сгенерировал модуль в процессе своей работы, можно прочитать в системном журнале /var/log/messages (обычно он доступен для чтения только администратором, т. к. содержит информацию, общий доступ к которой потенциально может повысить уязвимость системы). Кроме того, как упоминалось выше, можно посмотреть свежие сообщения ядра утилитой **dmesg.**

1.3 Комбинирование исходного кода на C и ассемблере

В приведенном простейшем модуле ядра не использовался доступ к аппаратной подсистеме компьютера (потому что этот модуль не делал никакой полезной работы). Однако реальным модулям ядра часто приходится взаимодействовать с аппаратурой (фактически, большая часть модулей ядра – это драйвера устройств). Поэтому при написании модулей актуальна задача выполнения машинных инструкций для взаимодействия с оборудованием.

Компилятор GNU C позволяет добавить к программе фрагменты ассемблерного кода несколькими способами. Можно отдельно скомпилировать ассемблерную процедуру в объектный файл, чтобы обратиться к ней из C-

программы, а можно вставить ассемблерный код прямо в текст программы на языке C в виде коротких `asm`-вставок.

Ассемблерные вставки в C-программе

В простейшем случае для вставки ассемблерного кода непосредственно в C-файл используется ключевое слово `asm`, после которого в круглых скобках указывается аргумент – строковый литерал с текстом ассемблерной вставки. Если ассемблерная вставка содержит несколько инструкций, их отделяют друг от друга так же, как это было бы сделано в обычной ассемблерной программе: переводом строки (\n) или переводом строки и символом табуляции (\n\t). Ассемблерная вставка может содержать любые ассемблерные инструкции, включая условный переход (в рамках этой же ассемблерной вставки) а также некоторые директивы, понятные ассемблеру as (GNU assembler из состава GCC).

Одна из особенностей ассемблера `as` – то, что он использует синтаксис AT&T, а не Intel. Особенности синтаксиса AT&T:

противоположный (по сравнению с синтаксисом Intel) порядок аргументов: например, `mov` *`источник, приёмник`*;

префиксы операндов (% перед именем регистра, $ перед непосредственным операндом и т. д.);

суффиксы команд, указывающие разрядность операции (`b` – байт, `w` – 2 байта, `l` — 4 байта и т. д.): например, `movl %ebx, %eax`.

Синтаксис AT&T может выглядеть понятнее, когда текстовый редактор не поддерживает подсветку синтаксиса; однако в остальных случаях его не без оснований считают избыточным. Но исторически именно он является стандартом в мире Unix, и в том числе использован в ассемблерных вставках ядра Linux.

Все это не значит, что синтаксис Intel невозможно использовать в собственном модуле ядра. Для переключения `as` на использование синтаксиса Intel требуется всего лишь добавить в ассемблерный код следующую директиву:

```
.intel_syntax noprefix
```

При этом будет не лишним вернуть синтаксис AT&T в конце ассемблерной вставки следующей директивой:

```
.att_syntax prefix
```

Обмен информацией между ассемблерной вставкой и основной программой можно выполнять несколькими способами. Самый простой — обращаться к объявленной в основной программе глобальной переменной (т. е. такой переменной, которая объявлена вне тела функции):

```
#include <linux/module.h>
#include <linux/kernel.h>

int hello1_variable;

int f(void)
{    hello1_variable=0;
     asm( ".intel_syntax noprefix\n"
          "mov DWORD PTR [hello1_variable], 100\n"
          ".att_syntax prefix");
     ....
```

ПРИМЕЧАНИЕ: в приведенном примере можно заметить небольшое отличие синтаксиса ассемблера gas от более привычного ассемблера NASM. Когда мы указываем команде mov размер аргумента мы пишем BYTE PTR для байтных аргументов, WORD PTR для словных, и т. д. В ассемблере NASM мы писали бы просто BYTE, WORD и т. д.

Работа с портами ввода-вывода

Большинство инструкций процессора включая инструкцию MOV, имеют доступ только к операндам в пространстве адресов памяти. Обращаться к портам ввода-вывода могут только две специальные инструкции – IN и OUT.

Инструкция IN копирует содержимое из указанного порта ввода-вывода в регистр AL или AX. Адрес порта ввода-вывода, указываемый в качестве источника, можно выбрать одним из двух способов. Если адрес порта меньше 256 (100h), вы можете указать его в инструкции, например:

```
in   al, 41h
```

Эта инструкция копирует байт из порта ввода-вывода 41h в регистр AL.

При втором способе вы можете использовать для ссылки на порт ввода-вывода, из которого нужно выполнить чтение, регистр DX:

```
...
mov   dx, 41h
```

```
in    al, dx
...
```

Инструкция OUT в точности эквивалентна инструкции IN, только операндом-источником является регистр AL или AX, а порт ввода-вывода, на который указывает регистр DX или постоянное значение, меньшее 256, является операндом-приемником. Например, следующие инструкции устанавливают порт ввода-вывода 3B4h в значение 0Fh:

```
...
mov   dx, 3b4h
mov   al, 0fh
out   dx, al
...
```

Макросы C для работы с портами ввода-вывода

Для некоторых распространенных инструкций заранее определены готовые встроенные функции. Например для работы с портами ввода-вывода можно использовать функции outb/outw (для вывода) и inb/inw (для ввода).

Функции outb/outw принимают в качестве первого аргумента байтное/двухбайтное значение, и выводят его в порт, заданный вторым аргументом. Функции inb/inw принимают единственный аргумент – номер порта ввода-вывода, из которого нужно прочитать байтное/двухбайтное значение. Прочитанные из порта данные являются возвращаемым значением функции. Понятно, что суффикс b в имени функции означает работу с байтными портами, суффикс w — с портами размером 2 байта. Пример их использования приведен ниже:

```
include <linux/io.h>
inline void cli() {
     asm("cli");
}
inline void sti() {
     asm("sti");
}
int readPort(int reg)
{
     outb(reg, 0x70);
     return inb(0x71);
}
     ....
     char value;
     cli();
     value = readPort(0x14)
```

```
sti();
....
```

В приведенном примере описана функция readPort(), выполняющая следующие действия:

- вывод переданного функции значения в порт 0x70;
- чтение значения из порта 0x71;
- возврат прочитанного значения в вызывающую программу.

Эта последовательность действий может казаться искусственной, но на самом деле такой способ обмена информацией является типичным. Контроллеры устройств компьютера обычно имеют куда больше внутренних регистров (или ячеек встроенной памяти), чем портов ввода-вывода. По этой причине сначала контроллеру через специальный порт ввода-вывода посылается байт или слово, указывающее, в какую именно внутреннюю ячейку мы хотим получить доступ, а затем через другой (соседний) порт осуществляется доступ к этой ячейке. В частности, приведенный пример — это доступ к одной из ячеек системной области NVRAM, хранящей начальную конфигурацию оборудования.

1.4 Чтение и запись системной области NVRAM (CMOS)

Получать доступ к защищенным областям памяти (пространству портов ввода-вывода) в данной работе мы будем на примере такой аппаратной подсистемы, неправильное обращение к которой наименее опасно для работы ОС, не должно вызвать серьезные сбои и перезагрузку компьютера. Такой подсистемой является память NVRAM (non-volatile RAM или долговременная память), хранящая настройки BIOS и исторически известная под аббревиатурой CMOS (от технологии КМОП, на базе которой она была построена в ранних персональных компьютерах). Типичный размер этой области памяти — 64 или 128 байт; он может отличаться в зависимости от модели материнской платы и чипсета, но обычно не превышает 512 байт).

Так же как и регистры микроконтроллеров всех устройств компьютера, физически CMOS не находится в ОЗУ. На современных материнских платах она обычно интегрирована в микросхему южного моста вместе с часами реального времени (RTC clock). Поэтому данные из CMOS нельзя прочитать прямым доступом к памяти, и приходится обращаться к адресному пространству портов ввода-вывода. Доступ к CMOS осуществляется через порты ввода/вывода 70h и 71h.

Исходя из строения порта 70h, CMOS память имеет верхний предел в 128 байт. Это связано с тем, что только 7 бит (0-6) используются для адресации и последний бит для включения/выключения немаскируемых прерываний (Non-Masked Interrupts, NMI) 1 — включить NMI, 0 — выключить NMI.

Для доступа к CMOS в порт 70h записывается индексный адрес байта (0-7Fh) и после этого (с небольшой задержкой) можно считывать содержимое указанного байта из порта 71h. Во время этих операций записи и считывания должны быть запрещены прерывания, т. к. в промежутке между записью в 70-ый порт и считыванием из 71-го может произойти прерывание, которое перезапишет значение в 70-ом порту. Запрет прерываний осуществляется сбросом флага IF (ассемблерной командой CLI). Разрешить прерывания можно, установив этот же флаг командой STI.

Карта памяти типичной области CMOS

До 128-го байта располагается CMOS: 16 байт (00h-0fh) — это RTC, 32 байта (10h-2Fh) — информация по настройке устройств шины ISA, 16 байт (30h-3Fh) — специфическая информация BIOS, 64 байта (40h-7Fh) — это ESCD (Extended System Configuration Data — расширенная информация о конфигурации системы).

В таблицах 1 и 2 приведено описание значений, хранящихся в CMOS.
Все значения RTC (Таблица 1) хранятся в BCD формате как 2 полубайта, но в десятичном формате. Например, 31 (dec) хранится как 31 (hex).

Таблица 1

Ад (HEX)	Описан бай
	Текущая секунда на часах реального времени
	Секунда для включения компьютера по будильнику
	Текущая минута на
	Минута для включения компьютера по будильнику
	Текущий час на часах
	Час для включения компьютера по будильнику
	Текущий день недели (1 Воскресень
	Текущий день месяца
	Текущий
	Текущий год только последние цифры напр.

Таблица 2

Адрес (HEX)	Описание
...	
0BH	Регистр статуса RTC (# B): Бит 0 - Режим летнего времени (1 = летнее время включено, 0 = отключено); Бит 1 — формат времени (если 0 то 12-часовой; если 1 то 24-часовой); Бит 2 - Режим даты (1 = дата в двоичном формате, 0 = дата в формате BCD, по умолчанию = 0); Бит 5 - прерывание будильника (0=отключено, по умолчанию = 0)
...	
12H	Тип винчестера. Биты 0-3: первый винчестер; Биты 4 -7: второй винчестер. Варианты значений: 0000 = диск не установлен; другое_значение = тип диска в понимании BIOS; 1111 = признак того, что актуальная информация о дисках хранится по адресам 19H и 1AH
13H	РЕЗЕРВ (может содержать техническую информацию на усмотрение производителя BIOS)
14H	Байт оборудования: Бит 0 = 1, если присутствует дисковод гибких дисков; Бит 1 = 1, если присутствует блок процессора для операций с плавающей точкой (FPU); Бит 2 = 1 если есть клавиатура; Бит 3 =1 если есть дисплей; Биты 5, 4 - основной видеоадаптер: · 00 - EGA или VGA, · 01 или 10 — CGA, · 11 — текстовый монохромный дисплей; Биты 6, 7 - количество дисководов гибких дисков (00=1, 01=2, 10=3, 11=4)
15H, 16H	Объем базовой памяти (в первом мегабайте ОЗУ) 15H - младший байт 16H - старший байт (в современных системах хранит значение 280H = 640K)
17H, 18H	Объем дополнительной памяти в Кб(в адресном пространстве за первым мегабайтом ОЗУ): 17 H - младший байт, 18 H - старший байт
19H	Тип жесткого диска № 0, в современных системах — 47 или 49, что означает «пользовательский тип» (user defined)
20H	Тип жесткого диска № 1, в современных системах — 47 или 49, что означает «пользовательский тип» (user defined)
...	
32H	Век в формате BCD
...	

1.5 Подробные сведения о системе

Как можно заметить, конфигурация системы, хранящаяся в NVRAM, минимальна и касается только тех узлов компьютера, которые необходимы для его начальной загрузки. Все остальные аппаратные подсистемы инициализируются и настраиваются ОС в процессе загрузки. По этой причине

ОС (после завершения процесса загрузки) обладает гораздо более полной информацией об аппаратуре компьютера, чем BIOS. Разные ОС предоставляют программисту разные способы получения этой информации. В Linux поддерживается несколько таких способов, и самый простой из них – виртуальная файловая система `proc`.

Специальная директория `/proc`, имеющаяся на каждой Linux-системе — это виртуальная файловая система, которая предоставляет доступ к информации ядра. Различная информация, которую ядро может сообщить пользователям, находится в «файлах» каталога `/proc`.

Файлы, содержащиеся в `/proc`, — действительно виртуальные. На самом деле, они не существуют на диске; операционная система создает их «на лету», если вы делаете попытку их прочитать.

Директория `/proc` состоит из виртуальных директорий и поддиректорий, которые группируют файлы по определенному принципу. Команда `ls /proc` выдаст что-то вроде этого:

```
1     2432  3340  3715  3762  5441  815
129   2474  3358  3716  3764  5445
1290  248   3413  3717  3812  5459
133   2486  3435  3718  3813  5479
1420  89    3439  3728  3814  557
165   276   3450  3731  39    5842
166   280   36    3733  3973  5854
2     2812  3602  3734  4     6
2267  3     3603  3735  40    6381
2267  26    3614  3737  4083  6558
2282  327   3696  3739  4868  6561
2285  3284  3697  3742  4873  6961
2295  329   3700  3744  4878  7206
2335  95    3701  3745  5     7207
2400  330   3706  3747  5109  7222
2401  3318  3709  3749  5112  7225
2427  3329  3710  3751  541   7244
2428  3336  3714  3753  5440  752
devices    modules
acpi       diskstats    mounts
asound     dma          mtrr
bus        execdomains  partitions
dri        fb           self
driver     filesystems  slabinfo
fs         interrupts   splash
ide        iomem        stat
irq        ioports      swaps
net        kallsyms     sysrq-trigger
```

```
scsi        kcore         timer_list
sys         keys          timer_stats
sysvipc     key-users     uptime
tty         kmsg          version
buddyinfo   loadavg       vmcore
cmdline     locks         vmstat
config.gz   meminfo       zoneinfo
cpuinfo     misc
```

Пронумерованные директории соответствуют каждому запущенному процессу; специальная символьная ссылка `self` указывает на текущий процесс. Некоторые виртуальные файлы предоставляют информацию об аппаратном обеспечении, например, `/proc/cpuinfo`, `/proc/meminfo` и `/proc/interrupts`. Другие дают информацию, связанную с файлами, например, `/proc/filesystems` или `/proc/partitions`. Файлы в `/proc/sys` относятся к конфигурации параметров ядра.

Команда `cat /proc/meminfo` может дать следующее:

```
# cat /proc/meminfo
MemTotal:       483488 kB
MemFree:          9348 kB
Buffers:          6796 kB
Cached:         168292 kB
```

Остановимся на некоторых файлах и каталогах:

/proc/acpi: директория, дающая информацию о системе управления питанием ACPI. К примеру, чтобы узнать, подключен ли ваш ноутбук к питанию AC, вы можете выполнить cat /proc/acpi/ac_adapter/AC/state и получить либо «on line», либо «off line»

/proc/cmdline: Показывает параметры, переданные ядру при загрузке.

/proc/cpuinfo: Дает данные о процессоре компьютера.

/proc/loadavg: Файл, показывающий среднюю загрузку процессора; его информация включает использование ЦПУ в последнюю минуту, последние пять минут и последние 10 минут, а также число процессов, работающих в данный момент.

/proc/stat: Также предоставляет статистику, но последней загрузки.

/proc/uptime: Короткий файл, в котором только два числа — сколько секунд ваша система работает и сколько секунд она простаивает.

/proc/devices: Показывает все настроенные и загруженные символьные и блочные устройства. **/proc/ide** и **/proc/scsi** дает данные об устройствах IDE и SATA.

/proc/ioports: Показывает информацию о задействованных устройствами портах ввода-вывода.

/proc/dma: Показывает используемые каналы прямого доступа к памяти.

/proc/filesystems: Показывает типы файловых систем, поддерживаемых ядром.

/proc/mounts: Показывает всё смонтированное, что используется компьютером (его вывод очень похож на /etc/mtab); /proc/partitions и /proc/swaps показывает все разделы и пространство файла подкачки (swap).

/proc/net: Содержит сетевую информацию. Описание каждого из файлов в этой директории заняло бы слишком много места, но в ней есть /dev (каждое сетевое устройство), сетевая статистика и статистика портов, информация о беспроводных подключениях и т. д.

Есть также несколько файлов, связанных с ОЗУ: /proc/iomem показывает, сколько памяти использует система, а /proc/kcore демонстрирует физическую оперативную память системы. Размер /proc/kcore соответствует размеру оперативной памяти компьютера. Наконец, здесь же есть много файлов и директорий, связанных с аппаратным обеспечением, такие как /proc/interrupts и /proc/irq (информация о прерываниях), /proc/pci (все устройства PCI), /proc/bus и т. д.

Задание для выполнения:

1. Изучить особенности создания, компиляции и загрузки модулей ядра Linux.
2. Написать простейший модуль ядра, выводящий сообщения при загрузке и выгрузке. Откомпилировать модуль и загрузить его в память. Проверить, загружен ли модуль в память, и убедиться в наличии выведенных им сообщений.
3. Модифицировать модуль, чтобы он читал информацию из системной области NVRAM (CMOS) и выводил закодированные в ней сведения об установленном оборудовании на экран в текстовом виде. Для чтения области NVRAM использовать ассемблерные вставки.

4. Повторить задание из предыдущего пункта, используя функции C для работы с портами ввода-вывода.

5. Проанализировать содержимое файлов подкаталога /proc; найти в них информацию об установленном оборудовании; найти в /proc какую-либо информацию, которую можно сравнить с данными, хранящимися в NVRAM (например, размер установленной оперативной памяти) и выполнить такое сравнение. Описать в отчете исследованные файлы и привести фрагменты их содержимого.

Контрольные вопросы

1. Объясните, как реализована аппаратная защита памяти в современных компьютерах

2. Что такое модули ядра, зачем они нужны (вообще и конкретно в данной работе)?

3. Как выполнить загрузку модуля ядра в память?

4. Назовите два уровня абстракции для доступа к портам ввода-вывода.

5. Какая информация хранится в области NVRAM (CMOS)?

6. Расскажите об общей структуре виртуальной файловой системы proc.

ЛАБОРАТОРНАЯРАБОТА №3
«ПЕРЕОПРЕДЕЛЕНИЕ КОДОВ КАРТЫ СИМВОЛОВ КЛАВИАТУРЫ»

2.1 Теоретические сведения

Таблица символов

Когда вы нажимаете клавишу на клавиатуре, ядро ОС генерирует код, который затем используется в качестве адреса в таблице кодов клавиш, чтобы выбрать из нее нужный символ. Манипуляции с этой таблицей позволяют работать, используя различные раскладки клавиатуры, установленный язык системы и экранные шрифты в различных кодировках.

В GNU/Linux предусмотрен специальный механизм, позволяющий вмешиваться в этот процесс и на лету изменять правила перекодировки вводимых символов.

Таблица, используемая для перекодировки кодов, поступающих с клавиатуры, называется картой символов — keymaps. В подкаталогах каталога

/usr/share/kbd/keymaps можно найти множество текстовых файлов с готовыми картами символов, предназначенных для различных клавиатурных раскладок и кодировок. Хранение карт символов в загружаемых файлах (так называемых map-файлах) позволяет пользователю самостоятельно менять раскладку клавиатуры редактированием соответствующего файла обычным текстовым редактором и последующей загрузкой измененного файла в память с помощью специальной утилиты.

Формат map-файла

В начале каждого файла обычно расположен комментарий, поясняющий назначение карты символов и особенности ее использования. Комментарием считается последовательность символов, начинающаяся с «!» или «#» и продолжающаяся до конца строки.

Синтаксис файлов символьных карт является построчным. Это значит, что полное описание переназначения клавиши должно вмещаться в одну строку. Если нужно перенести часть описания на новую строку, его можно разделить на несколько физических строк, но при этом каждая подстрока должна заканчиваться символом обратного слэша «\».

Файл может содержать в себе включения других файлов карт с помощью директивы

```
include "путь_к_подключаемому_файлу"
```

Указать кодировку символов можно командой charset:

```
charset "iso-8859-x"
```

Это задает, как будут интерпретированы последующие символы. Например, в кодировке iso-8859-1 символ «*μ*» имеет код 0265, а в кодировке iso-8859-7 — 0354.

Строка, содержащая полное описание клавиши, начинается с ключевого слова keykode и имеет вид:

```
keykode n = keysym keysym keysym...
```

где n (номер клавиши) — это внутренний идентификационный номер клавиши (упрощенно можно считать его эквивалентом скан кода). Он может быть задан в десятичном, восьмеричном или шестнадцатеричном виде. Восьмеричный номер должен начинаться с символа «0» (ноль), а шестнадцатеричный с префикса «0x».

Каждый указанный в строке keysym (символьный код) представляет собой *действие клавиатуры*. К одной клавише может быть привязано до 256 действий. Доступные действия включают вывод кода символа, вывод символьной последовательности, переключение консолей или раскладок клавиатуры, перезагрузку компьютера и т. д. Полный список действий (а главное, их обозначений) можно получить командой dumpkeys -l

Примечание: команду нужно запускать из текстовой консоли, а не из графической программы-терминала. Учитывая, что ее вывод очень объемный, луче перенаправлять его на экранный пейджер: например «**dumpkeys -l | less**».

Примечание к примечанию: напомним, что «текстовая консоль» — это текстовый экран с черным фоном, который появляется при нажатии одной из комбинаций клавиш **Control+Alt+F1** . . . **Control+Alt+F6** (соответственно переключение в 1-ю . . . 6-ю текстовые консоли). Текстовых консолей в большинстве дистрибутивов именно 6, а 7-я консоль (**Control+Alt+F7**) отводится под графическую оболочку. Когда мы запускаем из графической оболочки программу-терминал, в большинстве случаев она работает как текстовая консоль, и это удобно, т. к. на одном графическом экране можно запустить сразу несколько окон-терминалов. Однако некоторые особенности работы такого терминала диктуются графической оболочкой: например графическая оболочка берет на себя заботу о переключении раскладок клавиатуры. Поэтому для экспериментов с «чистой» консолью лучше переключаться в текстовый режим.

Какое действие будет производить нажатие клавиши, зависит от действующих в данный момент клавиш-модификаторов. Драйвер клавиатуры поддерживает 8 модификаторов: **Shift**, **AltGr**, **Control**, **Alt**, **ShiftL**, **ShiftR**, **CtrlL** и **CtrlR**. Каждый из этих модификаторов имеет свой вес, равный какой-либо степени двойки.

Эффективное действие клавиши можно определить, сложив веса действующих в момент ее нажатия модификаторов. По умолчанию ни один из модификаторов не воздействует на клавишу, поэтому эффективное действие равно 0 (при нажатии или отпускании клавиши выбирается самый первый keysym в строке).

Можно задавать keysym различными способами: численно — в десятичной, восьмеричной, шестнадцатеричной системах счисления — или в

кодировке Unicode, или в символьном виде. Численное обозначение аналогично заданию keynumber. Описание в кодировке Unicode начинается с последовательности «U+», за которой следуют четыре шестнадцатеричных цифры.

Если перед keysym стоит «+» (знак «плюс»), этот keysym трактуется как «буква» и потому будет подвержен влиянию модификаторов — **CapsLock** и **Shift**. ASCII-символы (a-z и A-Z) подвержены воздействию **CapsLock** по умолчанию. Будучи активированы одновременно, CapsLock и Shift отменяют друг друга, приводя к набору символа в нижнем регистре. Если сочетание **Shift+CapsLock+буква** не должно выдавать символ в нижнем регистре, в map-файле нужно убрать знак «плюс» перед заглавной буквой:

keycode 30 = +a A

Карты символов позволяют также определить одно единственное действие на конкретную комбинацию клавиш. Синтаксис такого определения выглядит следующим образом:

```
{ plain | <модификаторы> } keycode keynumber = keysym
```

Это особенно удобно, когда вы пользуетесь картой символов, не удовлетворяющей вас только несколькими сочетаниями клавиш — например, для переключения раскладки. Вы можете создать небольшой локальный файл, переопределяющий только нужные вам сочетания клавиш, а затем загрузить его в память вслед за основным файлом (ниже будет описано, как это сделать). Пример такого одиночного определения:

```
plain keycode 14 =
BackSpace control alt
keycode 83 = Boot alt
keycode 105 =
Decr_Console alt keycode
106 = Incr_Console
```

Заметим, что в приведенном примере присутствует несколько имен комбинаций клавиш, о назначении которых несложно догадаться: так, Boot соответствует перезагрузке, Decr_Console — переключению на предыдущую текстовую консоль, а Incr_Console — на следующую.

Ключевое слово plain определяет только базовый код клавиши (когда не действует ни один модификатор), не затрагивая действия, привязанные к остальным сочетаниям модификаторов с данной клавишей.

Объявление строк

Вдобавок к комментариям и объявлениям кодов клавиш, карта символов может содержать объявления строк:

```
string keysym = "text"
```

Текст может содержать буквы, восьмеричные коды (в формате обратной косой черты, за которым следует не более трех восьмеричных цифр) и три escape-последовательности «\n», «\\», и «\"», соответственно для перехода на новую строку, вывода обратной косой черты и кавычки.

Например, можно сделать так, чтобы клавиша **F5** (расположена на клавиатуре вверху) выводила при нажатии на нее текст «Hello!», клавиша **Shift+F5** — «Good bye!» (используем для этого заведомо незадействованные keysym):

```
keycode 63 = F70 F71
string F70 = "Hello!"
string F71 = "Goodbye!"
```

Напомним, что для того, чтобы узнать, какие keysym доступны для использования в картах символов, используется команда **dumpkeys -l**.

Примеры использования

Пример ниже меняет местами левый **Control** и **CapsLock**:

```
keycode 58 = Control
keycode 29 = Caps_Lock
```

Следующий пример включает набор клавиш расширенной клавиатуры:

```
keycode 102 = Insert
keycode 104 = Remove
keycode 107 = Prior
shift keycode 107 = Scroll_Backward
keycode 110 = Find
keycode 111 = Select
control alt    keycode 111 = Boot
```

```
control altgr keycode 111 = Boot
```

Последний пример показывает, как привязать строку «du\ndf\n» к сочетанию **AltGr-D**. При этом мы используем «незанятый код» F100, обычно не привязанный к какому-либо действию:

```
altgr keycode 32 = F100
string F100 = "du\ndf\n"
```

Изменение и загрузка файла keymap

Изменение карты символов

Для создания своей раскладки проще всего отредактировать файл, содержащий похожую карту символов (например, можно взять за основу украинскую раскладку в кодировке Unicode с переключением кириллицы правой клавишей **Control**, находящуюся в файле **/usr/share/keymaps/i386/qwerty/ua-utf.kmap.gz**). *Скопировав тарфайл в свой домашний каталог*, его можно отредактировать, например, *редактором* ***mcedit.*** Код нужной клавиши можно получить программами **showkey** (следит за нажатиями) и **dumpkeys** (выводит все сразу).

Допустим надо, чтобы клавиша **Backspace** отсылала код **Backspace** (**control-H**), а не **Delete** (**control-?**):

```
keycode 14 = BackSpace
```

Можно или изменить существующую строку, или добавить новую. Или, например, для замены действия клавиши **CapsLock**, найдите строчку: «keycode 58 = Caps_Lock» и замените ее на «keycode 58 = Control», после чего сохраните файл под тем же именем, или под другим, например, **yourname.kmap.gz.**

Загрузка карты символов утилитой loadkeys

Загрузить новый map-файл (в текстовой консоли) можно командой **loadkeys**, выполнив в консоли:

```
sudo loadkeys ./filename.kmap.gz
```

Программа **loadkeys** загружает трансляционные таблицы или их части. Синтаксис программы в общем случае имеет вид:

```
loadkeys [ -c --clearcompose ] [ -C '<cons1 cons2 ...>'
    | --console=cons1,cons2,... ] [ -d --default ]
[ -h --help ] [ -m --mktable ] [ -s --clearstrings ]
[ -v --verbose ] [ filename... ]
```

Программа читает файл с именем **filename** и загружает карту символов ядра в консоль. Консоли, на которые должна повлиять команда, можно задать явно, параметром `-C (--console)`. Этот параметр распознает список имен устройств (консолям с 1-й по 6-ю обычно соответствуют устройства **/dev/tty1 . . . /dev/tty6**). Если ничего не указано, команда действует на текущую консоль.

Если передан параметр `-d (--default)`, программа загрузит стандартную карту символов (вероятнее всего **defkeymap.map** в **/usr/share/kbd/keymaps** или в **/usr/src/linux/drivers/char**).

Загрузка карты символов ядра — это основная функция программы. Если задано имя файла, таблица читается из него, в противном случае карта читается из стандартного потока ввода.

Утилиты getkeycodes и setkeycodes

Программа **getkeycodes** выводит карту, связывающую сканкод (код, аппаратно генерируемый клавиатурой) и keykode (код_клавиши). Программа не принимает параметров и аргументов.

Программа **setkeycodes** загружает таблицу, связывающую сканкоды с keykod'ами. Синтаксис программы прост:

```
setkeycodes сканкод keykode ...
```

Программа читает по два аргумента за раз, каждая пара содержит скан код (**в шестнадцатеричном виде**) и keykode (**в десятичном**). Для переназначения кодов следует запускать программу через **sudo.**

Управление раскладкой в графической оболочке

Графическая среда X Window System берет на себя отображение кодов клавиш на конкретные keysym. В целом ситуация аналогична использованию map-файлов и **loadkeys**, но содержит в себе некоторые отличия и особенности.

Действия клавиатуры также представляются строкой вида

```
keycode n = keysym1 keysym2 keysym3 keysym4 ...
```

В этой строке i-й keysym также соответствует конкретному активированному модификатору. Поддерживаются следующие 6 вариантов:

1. клавиша без модификаторов

2. **shift** + клавиша
3. **mode_switch** + клавиша
4. **mode_switch** + **shift** + клавиша
5. **AltGr** + клавиша
6. **AltGr** + **shift** + клавиша

Не обязательно устанавливать все keysym. Завершающие незначащие позиции можно просто не указывать, а для указания незначащей (не вызывающей никакого действия) промежуточной позиции используется ключевое слово NoSymbol. Комментарии также обозначаются восклицательным знаком.

Для работы с картами символов среды X используется утилита **xmodmap**. Она позволяет просматривать и модифицировать действия клавиатуры. Синтаксис программы:

```
xmodmap [-параметры ...] [имя файла]
```

Перечислим некоторые полезные параметры команды. Параметр «-n» позволяет посмотреть, что будет делать **xmodmap** без внесения изменений в карты символов. Параметр «-e выражение» указывает выражение для выполнения. Параметр «-pk» выводит текущую карту символов. Параметр «-pke» выводит текущую карту символов в формате выражений, которые принимает **xmodmap**. И, наконец, «-» (простой дефис) дает понять программе, что в качестве входного файла должен использоваться стандартный поток ввода.

Применение xmodmap

Сохранить действующую карту символов в текстовый файл можно, выполнив:

```
xmodmap -pke > ~/myxmap
```

Загрузить модифицированный файл карты можно, вызвав

```
xmodmap ~/myxmap
```

Поскольку для полноценного использования загружать map-файл в память приходится при каждом запуске X Window System, обычно его делают

скрытым (добавляют точку в первом символе имени), а вызов загружающей его команды прописывают в автозагрузку. Полновесные окружения рабочего стола, такие как GNOME и KDE, берут на себя эту задачу, предоставляя пользователю графический диалог настроек раскладки клавиатуры; однако внутри они используют описываемые механизмы X Window System.

Получить keykode (и другую информацию о клавише) можно с помощью утилиты **xev**, входящей в стандартный набор программ X Window System. При запуске **xev** из терминала открывается новое окно, и вся информация о событиях клавиатуры и мыши, относящихся к этому окну, выводится в терминал. Например, нажав клавишу A (событие KeyPress), а затем отпустив ее (KeyRelease), можно получить в терминале следующий текст:

```
KeyPress event, serial 26, synthetic NO, window 0x340000
1,
     root 0xae, subw 0x0, time 5800192, (600,303), root:(
604,326),
     state 0x12, keycode 38 (keysym 0x41, A), same_screen
YES,
     XLookupString gives 1 bytes: "A"
KeyRelease event, serial 26, synthetic NO, window 0x3400
001,
     root 0xae, subw 0x0, time 5800248, (600,303), root:(
604,326),
     state 0x12, keycode 38 (keysym 0x41, A), same_screen
YES,
     XLookupString gives 1 bytes: "A"
```

Приведем пример использования команды. Чтобы при нажатии на клавишу «l» выводился символ «e», а при нажатии на клавишу «L» — символ «E», нужно заменить «keycode 46 = l L l L lstroke Lstroke lstroke» на «keycode 46 = e E l L lstroke Lstroke lstroke» (в зависимости от вашей раскладки keykode может иметь и другой номер). Можно также подменить весь keysym: «keysym a = e E». Изменения приведут к аналогичным заменам, т.е. «a → e» и «shift+a

→ E».

Изменения для текущего сеанса можно проверять, запуская **xmodmap** из терминала с указанием непосредственного действия, а не файла с картой. Например:

```
xmodmap -e "keycode 46 = l L l L lstroke Lstroke lstroke"
xmodmap -e "keysym a = e E"
```

Рассмотрим еще несколько полезных команд. Команда «clear ИМЯ_МОДИФИКАТОРА» удаляет все значения для заданного модификатора. Команда «add ИМЯ_МОДИФИКАТОРА = ИМЯ_KEYSYM ...» добавляет всем клавишам, содержащим заданные keysym, указанный модификатор. Наконец, команда «remove ИМЯ_МОДИФИКАТОРА = ИМЯ_KEYSYM ...» удаляет все клавиши, содержащие заданные keysym, из заданной карты модификатора.

Благодаря этим командам вы можете переназначить клавиши **shift**, **ctrl**, **alt** и **super** (они всегда существуют в левом и правом варианте, **Alt_R=AltGr**, а **Super** — это клавиша, при нажатии на которую ОС Windows показывает меню «Пуск»). Прежде всего необходимо очистить действия для клавиш, подлежащих переназначению. Для этого в начале map-файла можно написать что-то типа:

```
clear Control clear Mod4
keycode   8 =
...
```

Пример очищает **Control** и **Mod4** (**Super_L Super_R**). Новые назначения можно записать в конце map-файла:

```
keycode 255 =
add Control = Super_L Super_R
add Mod4 = Control_L Control_R
```

В данном случае мы поменяли местами клавишу **Super** склавишей **Ctrl**.

Утилита **xmodmap** также позволяет переназначить клавиши мыши. Для этого используется ключевое слово pointer. Например, «**pointer = default**» сбрасывает номера клавиш мыши к стандартным настройкам (по умолчанию): левая клавиша генерирует код 1, средняя клавиша — код 2, правая — код 3 и т. д.

Посмотреть, какие номера имеют кнопки мыши в текущий момент можно утилитой **xev**. У стандартной мыши обычно присутствуют кнопки с 1-й по 3-ю и еще две виртуальные кнопки (4-я и 5-я) соответствуют направлениям вращения колеса прокрутки. Геймерские мыши позволяют задействовать больше кнопок и осей прокрутки, и их порядок можно произвольно поменять, выполнив инструкцию следующего вида:

```
xmodmap -e "pointer = 1 3 2 4 5"
```

В этом примере мы поменяли местами 2-ю и 3-ю кнопки.

2.2 Задание для выполнения

1. Переназначить действия, выполняемые при нажатии клавиши (выбирается преподавателем), с изменением событий при модификаторах **Shift**, **AltGr**, **Control**, **Control+Shift**, **Alt** и **Control+Alt** на заданные преподавателем, с использованием только карты символов.
2. Привязать к сочетанию клавиш (выбирается преподавателем) вывод строки (выбирается преподавателем). Загрузку можно осуществлять при помощи **loadkeys**.
3. Переназначить действия клавиш, выбранных преподавателем, при помощи **setkeycodes**.
4. Переназначить действия клавиш, выбранных преподавателем, при помощи **xmodmap** (при работе в системе X).

2.3 Контрольные вопросы

1. Что такое карты символов и какие строки они могут содержать?
2. Чем отличаются понятия «сканкод» и «keykod»?
3. Чем отличается представление символов в различных кодировках?
4. Назовите способы управления раскладкой в графической оболочке.
5. Как можно исследовать обработку событий клавиатуры и мыши с помощью утилиты **xev**?

ЛАБОРАТОРНАЯ РАБОТА № 4
«РАБОТА С КОНТРОЛЛЕРОМ КЛАВИАТУРЫ ПО ПРЕРЫВАНИЮ ТАЙМЕРА»

1.Теоретические сведения

Общие сведения о cron

В операционной системе UNIX и ей подобных для запуска программ по расписанию используется механизм, называемый cron. Пользователь, желающий запускать свои программы по расписанию, создает конфигурационный файл, куда записывает расписание запуска программ. Данный конфигурационный файл каждую минуту просматривается cron'ом и запускаются те программы, время запуска которых подошло.

Редактируется конфигурационный файл с помощью программы crontab. Программы, запускаемые cron, исполняются от имени пользователя – владельца соответствующего конфигурационного файла.

Основной файл конфигурации cron, /etc/crontab, выглядит примерно так:

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.monthly )
#
```

1.1 crontab

Каждый пользователь системы имеет свой файл заданий crontab, в котором описано, в какое время и какие программы запускать от имени этого пользователя. Программа crontab позволяет редактировать файл заданий crontab, не прерывая процесс cron на время редактирования.

Редактирование crontab

Редактировать конфигурационный файл можно двумя способами.

Способ первый:

Суть его заключается в прямом редактировании через crontab.

- Наберите в командной строке команду crontab -e. Откроется редактор (возможно, после диалога выбора редактора) с содержимым вашего конфигурационного файла.

Если это первый запуск crontab'a, то файл будет пустой.

- Отредактируйте содержимое.

- Выйдите из редактора. Crontab автоматически даст команду демону cron перечитать Ваш файл.

Способ второй:

Заключается в приказе демону (системной программе, работающей в фоновом режиме) cron прочитать список заданий из указанного текстового файла.

- Создайте какой-нибудь текстовый файл в привычном для Вас редакторе.

- Выполните команду crontab имя_файла, где в качестве имени файла указан ваш свежесозданный файл. Crontab при этом даст команду демону cron прочитать Ваш файл.

Не следует путать конфигурационный файл crontab с тем текстовым файлом, который создаете Вы сами. Команда crontab имя_файла копирует содержимое указанного файла в специальный конфигурационный файл crontab, после этого Ваш текстовый файл в процессе больше не участвует.

Структура конфигурационного файла cron

Конфигурационный файл состоит из строк, каждая из которых описывает программу, которая будет запускаться по расписанию. Каждая строка состоит из 6 полей, поля отделяются друг от друга пробелом или табуляцией. Поля имеют следующее назначение:

1. Минуты (0-59)
2. Часы (0-23)
3. День месяца (1-31)
4. Месяц в году (1-12)
5. День недели (0-6, при этом 0 означает воскресенье)
6. Программа, которая будет запущена

Каждое из первых 5 полей может быть записано несколькими способами:

1. Символом * (означает любое значение)
2. Списком через запятую (1,2,3)
3. Диапазоном через тире (1-31)
4. Шагом значений диапазона (например */2 означает через_раз)

Кроме того, Вы можете получать по почте результаты и ошибки выполнения программ, запускаемых демоном cron по расписанию. Для этого в конфигурационном файле напишите MAILTO=ваш_email_адрес.

```
* * * * * выполняемая команда
- - - - -
| | | | |
| | | | ----- День недели  (0 - 7) (Воскресенье =0  =7)или
| | | ------- Месяц (1 - 12)
| | --------- День (1 - 31)
| ----------  (0 - 23)Час
------------- Минута (0 - 59)
```

Пример файла crontab

```
#  как обычно с символа,   '#' начинаются комментарии
#  в качестве командного интерпретатора использовать              /bin/sh
SHELL=/bin/sh
# результаты работы отправлять по этому адресу
MAILTO=paul@example.org
# добавить в  PATH
PATH=$PATH:$HOME/bin

#### Здесь начинаются задания

# выполнять каждый день в часов минут результат складывать в 05,log/daily
5 0 * * * $HOME/bin/daily.job >> $HOME/log/daily 2>&1
# выполнять числа каждого месяца в 1                    14 часов 15 минут
15 14 1 * * $HOME/bin/monthly
# каждый рабочий день в         22:00
0 22 * * 1-5 echo "Пора домой " | mail -s " 22:00" john Уже

23 */2 * * * echo "Выполняется в  0:23, 2:23, 4:23  . ."и т д

5 4 * * sun echo "Выполняется в  4:05  в воскресенье"
0 0 1 1 * echo " С новым годом      !"
15 10,13 * * 1,4 echo " Эта надпись выводится в понедельник и четверг в 10:15
и13:15"
0-59 * * * * echo "Выполняется ежеминутно      "
# каждые минут 5
*/5 * * * * echo "Прошло пять минут            "
```

Пример Crontab в Ubuntu

Редактируем от пользователя: crontab -e

Пример показывает ночной запуск программы скачивания (для экономии на льготных тарифах):

```
# m h  dom mon dow   command
# Запускаю eMule ночью в час ночи   1   10 минут
10 1  * * *  export DISPLAY=:0 && amule #
Останавливаю Emule утром в  10 часов 10 минут
10 10 * * *  export DISPLAY=:0 && killall amule
export DISPLAY=:0 && -Выводим на дисплей если есть что выводить         (
                                                    )
```

Примечание: когда через crontab (или просто из текстовой консоли) запускается графическая программа, ей необходимо установить переменную окружения DISPLAY в значение «:0» для для того, чтобы программа «нашла» X-сервер. Это можно сделать командой export.

Например, запуск редактора gedit можно выполнить из текстовой консоли, набрав в ней «export DISPLAY=:0 && gedit» (разумеется, чтобы увидеть запущенную программу, нужно переключиться в графическую консоль). Для получения дополнительной информации по этой теме можно воспользоваться встроенной справочной системой, набрав в командной строке сервера: man cron – описание особенностей функционирования cron; man 1 crontab – описание способов использования crontab; man 5 crontab – описание формата конфигурационных файлов для cron.

Пример инструкции для выполнения по cron

Для проверки работоспособности выполнения команд по расписанию мы будем использовать отложенный вывод в текстовую консоль через виртуальное устройство /dev/tty.

Устройства "tty" это аббревиатура от "Teletype". Первые терминалы были телетайпами (удаленно контролируемые устройства ввода). Файлы этих устройств находятся в подкаталоге /dev/ и пронумерованы (с запасом) для всех текстовых консолей. Кроме того, устройство /dev/tty (без номера) соответствует текущему терминалу или консоли, с которыми пользователь работает в данный момент. Таким образом, вывод на консоль можно осуществить так:

```
echo "hello" > /dev/tty
#посимвольно
```

```
echo "h\c" > /dev/tty echo
"e\c"> /dev/tty echo
"l\c"> /dev/tty echo
"l\c"> /dev/tty echo "o">
/dev/tty
```

Прикладная программа также может выводить в dev/tty, используя привычные в Си функции работы с файлами: fopen для открытия, fwrite для записи желаемых данных и fread для чтения, fclose.

Для того, чтобы прикладная программа могла писать в устройство /dev/tty0, необходимо предоставить пользователю права на доступ к соответствующему файлу. Текущую ситуацию с правами доступа можно узнать, выполнив
«ls -l /dev/tty0»:

```
$ ls -l /dev/tty0
```

crw--w---- 1 root tty 4, 0 нояб. 3 09:12 /dev/tty0

В приведенном примере полный доступ к файлу имеет пользователь root; кроме того, члены специальной группы tty имеют права на запись (но не чтение). В нашем случае для вывода сообщения в терминал достаточно прав на запись; однако, если пользователь, запускающий программу, не включен в группу tty — единственный способ ее успешного запуска — это запуск через команду sudo (на что пользователю тоже необходимы права). Обычно право запускать через sudo любую программу имеют пользователи из группы администраторов (например, владелец компьютера). Кроме того, непривилегированным пользователям можно разрешить запуск через sudo некоторых отдельных команд. Права на использование sudo настраиваются в файле /etc/sudoers. Например для данных лабораторных работ в /etc/sudoers разрешен запуск команд insmod и rmmod, благодаря чему становится возможным загружать и выгружать модули ядра, не имея привилегии администратора.

1.2 Работа с устройствами ввода-вывода из прикладной программы

В первой работе мы познакомились с доступом к портам ввода-вывода из пространства ядра: как с помощью ассемблерных инструкций (in и out), так и с помощью функций C (inb и outb). Необходимость написания модуля была связана с тем, что доступ к аппаратному обеспечению требует повышенных привилегий, а эти привилегии автоматически присущи коду ядра ОС.

В Linux существуют и другие способы доступа к аппаратному обеспечению, не требующие запуска собственного кода (модуля) в пространстве ядра. Вот два таких способа:

- эскалация привилегий прикладной программы с помощью вызова системной функции ioperm() или ей подобной; если вызов был успешным, прикладной программе становятся доступны низкоуровневые средства ввода-вывода (например, семейство функций inb/outb);
- использование файлов устройств в каталоге /dev – виртуальных файлов, на которые ядро отображает различные устройства компьютера; благодаря такой файловой абстракции для низкоуровневого доступа к устройствам можно использовать обычные средства файлового ввода-вывода C.

Следует учитывать, что оба способа требуют передачи прикладной программе повышенных привилегий. Файлы устройств обычно доступны только администратору (пользователю root), и таких же привилегий требует выполнение функции ioperm() . Поэтому чтобы любой из этих способов работал, программу нужно запускать через sudo (при наличии у пользователя соответствующих прав).

Низкоуровневый доступ из прикладной программы сопряжен с переключением в контекст ядра для выполнения каждой низкоуровневой операции. Переключение контекста — ресурсоемкая операция, этот способ общения с устройствами ввода-вывода заметно медленней, чем работа из модуля ядра. Однако иногда его удобно использовать в простых задачах, не требующих высокой производительности.

В качестве примера работы с файлами устройств рассмотрим использование файла /dev/port, на который ядро отображает все пространство портов ввода-вывода.

Доступ через файл портов ввода-вывода

Файл устройства /dev/port отображает все пространство портов ввода/вывода компьютера. Поскольку этот файл - очень мощное орудие, позволяющее производить любые манипуляции с аппаратурой, права на доступ к нему есть только у администратора системы. Смещение в этом файле соответствует номеру порта, и т.о. прочитав из файла байт по конкретному смещению, мы на самом деле прочтем его из порта с таким номером. Запись

байта (или слова) в порт выполняется записью его значения в файл /dev/port по соответствующему смещению.

Далее в этой работе, во избежание эскалации привилегий, мы будем взаимодействовать с портами через модуль ядра.

1.3 Работа с контроллером клавиатуры

Скан-коды

Процессор клавиатуры проводит большую часть времени за «сканированием» матрицы клавиш. Когда он обнаруживает, что какую-то клавишу нажали, отпустили, или удерживают в нажатом положении — он посылает в хост-устройство (компьютер) информацию, известную как скан-код. Скан-коды бывают двух типов: коды нажатия и коды отпускания клавиш. Каждой клавише присвоены уникальные коды нажатия и отпускания, а сканкоды всех клавиш клавиатуры называются набором сканкодов. Подчеркнем, что сканкод, генерируемый при нажатии или отпускании клавиши, представляет только эту клавишу — а не нарисованный на ней символ. Коды клавиш связываются с кодами конкретных символов уже в компьютере — например, в драйвере операционной системы.

Коды нажатия, отпускания и повтора клавиш:

- Большинство кодов нажатия клавиши — однобайтные, но несколько клавиш генерируют при нажатии двухбайтные или четырехбайтные коды (их называют расширенными клавишами).
- Большинство кодов отпускания клавиши — двухбайтные: первый байт у них F0h, а второй вычисляется по формуле «код_нажатия_данной_клавиши+ 080h».

Наборы скан-кодов и эта формула были придуманы для первых клавиатур, на которых было заметно меньше клавиш. Когда клавиш стало больше, одного байта перестало хватать. Для сохранения совместимости (включая формулу) пришлось придумать новым клавишам многобайтные коды. Поэтому коды нажатия и отпускания расширенных клавиш начинаются с байта E0h. У кода отпускания расширенной клавиши следом всегда идет байт F0h. Последний байт кода нажатия и отжатия расширенной клавиши конструируется по формуле, как в случае однобайтных кодов. Итого, код отжатия всегда на один байт длиннее

кода нажатия клавиши. Приведем несколько примеров для стандартного набора скан-кодов:

Клавиша	Код нажатия	Код отпускания
"A"	1C	F0,9C
"5"	2E	F0,AE
"F10"	09	F0,89
"→"	E0, 74	E0, F0, F4
правый "Ctrl"	E0, 14	E0, F0, 94

При удерживании клавиши в нажатом состоянии клавиатура будет (после небольшой паузы) циклически посылать код нажатия, пока не будет нажата какая-либо другая клавиша или пока не будет отпущена данная. Задержка перед циклической генерацией кода нажатия называется задержкой повторения клавиши и бывает от 0.25 до 1.00 секунд. Скорость повторения варьируется от 2.0 до 30.0 символов в секунду. Задержку/скорость повторения можно менять специальной командой контроллера клавиатуры.

ПРИМЕЧАНИЕ: Клавиша "Pause/Break" является исключением из всех правил: она имеет только код нажатия и не генерирует ничего при отпускании.

Команды контроллера клавиатуры

В процессе работы контроллер клавиатуры сканирует матрицу клавиш, передает скан-коды в компьютер, а также принимает от компьютера и выполняет некоторые служебные команды.

Есть два популярных способа подключения клавиатуры к компьютеру:

- USB keyboard — наиболее современные клавиатуры, подключаемые к универсальной последовательной шине, и их программирование мы здесь не рассматриваем.
- IBM-совместимые клавиатуры, известные также как "AT-клавиатуры" и "PS/2клавиатуры", поддерживаются всеми современными непортативными компьютерами.

Для подключения IBM-совместимых клавиатур к настольным компьютерам используется микроконтроллер, совместимый с микросхемой Intel 8042. В современных моделях он является частью чипсета на материнской плате, но

логически по-прежнему существует как самостоятельное устройство, известное как "i8042".

Для управления IBM-совместимой клавиатурой существует существует специальный набор команд контроллера i8042. Код команды подается в специальный регистр контроллера.

Сначала мы кратко перечислим основные команды (чтобы можно было составить представление о возможностях контроллера i8042). Затем рассмотрим подробнее несколько команд, имеющих отношение к текущей лабораторной работе, и то, как посылать их контроллеру.

- Команда EDh — изменить состояние светодиодов клавиатуры (следом за байтом команды идет байт данных, кодирующий, какие светодиоды должны гореть);
- Команда EEh — эхо-запрос. Клавиатура должна ответить, сгенерировав скан-код EEh;
- Команда F3h — Установить параметры режима автоповтора (следом за байтом команды идет байт данных, кодирующий скорость автоповтора сканкодов зажатой клавиши);
- Команда F4h — включить клавиатуру;
- Команда F5h — выключить клавиатуру;
- Команда F6h — установить параметры по умолчанию;
- Команда FEh — послать последний сканкод еще раз;
- Команда D2h — имитировать нажатие либо отпускание клавиши (следом за байтом команды идет сканкод, указывающий нужную клавишу);
- Команда FFh — выполнить сброс клавиатуры

Кроме перечисленных, есть еще команды, унаследованные от от устройств 80-х годов XX века. Некоторые современные реализации контроллера i8042 могут даже не отсылать эти устаревшие команды клавиатуре, а только делать вид, что они отправлены, и генерировать «ответ», как будто клавиатура их приняла. К числу таких устаревших команд, например, относятся:

- Команда F0h — выбрать кодовый набор (считается, что кроме набора скан-кодов поумолчанию клавиатура умеет генерировать еще несколько наборов сканкодов, давно никем не используемых);
- Команда A5h — установить защитный пароль, ограничивающий доступ к клавитуре

(потом этот пароль можно «запросить» у пользователя, послав клавитуре команду A6h, в результате чего клавиатура перестанет работать до правильного ввода пароля).

- Команды FBh, FCh, FDh — запретить для заданных клавиш генерацию кодов отпускания, автоповтора, либо того и другого сразу.

Управление контроллером клавиатуры

Контроллер i8042 имеет 4 однобайтных регистра:

- буфер ввода — регистр для чтения, содержит байт, считанный с клавиатуры;
- буфер вывода — регистр для записи, содержит байт, который должен быть записан в клавиатуру;
- регистр статуса — регистр для чтения, содержит байт с флагами состояния;
- регистр управления — регистр для чтения и записи, доступный только через специальную команду контроллера, содержит управляющие флаги.

В отличие от регистра управления, обращаться к первым трем регистрам (ввода, вывода и статуса) легче: они доступны напрямую через порты 0x60 и 0x64. Таблица показывает как используются эти периферийные порты. Поясним смысл таблицы. При чтении байта из порта 0x60 мы получаем сканкод нажатой клавиши, и в этот же порт мы записываем байт, который должен быть передан контроллеру клавиатуры (например, байт данных какой-либо команды). Иными словами, порт 0x60 соединен с двумя разными регистрами контроллера: буфером ввода (при чтении) и буфером ввода (при записи). При записи в порт 0x64 мы передаем команду контроллеру, а при чтении этот порт связывается с регистром статуса контроллера и передает его текущее значение.

Функция	R /W	Порт
0x60	Read	Чтение буфера ввода
0x60	Write	Запись буфера вывода
0x64	Read	Чтение регистра статуса
0x64	Write	Передача команды контроллеру

Благодаря этой хитрой схеме можно, имея всего два однобайтных порта, взаимодействовать с несколькими регистрами контроллера.

Порядок взаимодействия с контроллером выглядит так:

1. Прочитать содержимое регистра статуса из контроллера 0x64. Это позволяет узнать, готов ли контроллер к приему команд или данных, а также есть ли в буфере несчитанный сканкод (мы рассмотрим биты этого регистра чуть позже).
2. Запись в порт 0x64 не изменяет какой-либо регистр, но передает контроллеру i8042 команду, которую он должен выполнить. Если команда должна иметь параметр, этот параметр посылается в порт 0x60. Если команда возвращает какое-то значение — она помещает его в выходной буфер, и таким образом результат можно прочитать из порта 0x60.

Набор команд:

При получении команды клавиатура первым делом очищает выходной буфер. На неправильные команды (т. е. незнакомый код операции) контроллер отвечает байтом 0xFE ("resend") в выходном буфере. Если клавиатура ожидает байт аргумента команды, а вместо этого получает новую команду, прежняя команда теряется. Сканкоды в процессе обработки команды не генерируются.

Регистр статуса:

Флаги статуса 8042 читаются из порта 0x64. Они содержат информацию об ошибках, статусе, указывают, есть ли данные в буферах ввода и вывода:

бит 7: ошибка четности при передаче данных с клавиатуры

бит 6: тайм-аут при приеме бит 5: тайм-аут при передаче

бит 4: клавиатура закрыта ключом бит 3: данные, записанные в регистр ввода, — команда

бит 2: самотестирование закончено

бит 1: в буфере ввода есть данные (для контроллера клавиатуры)

бит 0: в буфере вывода есть данные (для компьютера)

Для чтения данных из клавиатуры используется порт 60h. При чтении из него можно получить скан-код последней нажатой клавиши:

```
in          al,60h;          прочитать скан-код клавиши,
cmp             al,hot_key;          если это не наша "горячая"
клавиша,          jne           not_our_key;    перейти на метку
not_our_key
   [...]                       ; наши действия здесь
not_our_key:
   [...]
```

Ассемблерные вставки в С-программе (расширенный синтаксис)

Как мы помним из первой работы, для вставки короткого ассемблерного кода непосредственно в С-файл используется ключевое слово asm, после которого в круглых скобках указывается обязательный аргумент – строковый литерал с текстом ассемблерной вставки. Кроме обязательного аргумента этой команде можно указать еще несколько необязательных аргументов, отделённых двоеточием:

```
asm( текст_ассемблерной_вставки
: список_выходных_операндов
: список_входных_операндов
: список_разрушаемых_регистров );
```

Для удобной связи ассемблерных инструкций с переменными программы используются два необязательных аргумента (списки входных и выходных операндов), в которых переменные С-программы перечислены через запятую. Обращение к такому операнду в тексте ассемблерной вставки осуществляется по номеру с префиксом %: %0, %1 и т. д. (нумерация в списках выходных и входных операндов сплошная).

Каждый операнд имеет следующий вид:

```
ограничение_типа (имя_переменной)
```

Здесь имя_переменной – это имя переменой в программе на C, а ограничение_типа – строковая константа. Эта строковая константа для выходных операндов должна начинаться с символа «=» (операнд только для записи). Следующий символ указывает, куда компилятору поместить значение переменной («r» – регистр, «m» – память и т. д.). Например, следующая ассемблерная вставка занесет в переменную i число 22:

```
int i;
asm( ".intel_syntax noprefix\n"
     "mov %0, 22\n"
     ".att_syntax prefix\n"
    :"=r"(i) );
```

Последний элемент оператора ассемблерной вставки, список_разрушаемых_регистров, сообщает компилятору о тех регистрах, которые будут «разрушены» в результате выполнения кода вставки. Заметим, что имена регистров в этом аргументе (через запятую) надо предварять знаком «%» (в соответствии с синтаксисом AT&T). Приведем более полный пример использования ассемблерной вставки:

```
int argument=10, rezultat;
asm( ".intel_syntax noprefix\n\t"
"mov eax, %1\n\t"
"mov %0, eax\n\t"
:"=r"(rezultat)
:"r"(argument)
:"%eax"
}
```

В первой инструкции ассемблерной вставки в регистр eax помещается значение переменной argument (которая указана в качестве входного операнда и потому доступна внутри ассемблерной вставки под именем %1). Во второй инструкции в переменную rezultat пересылается значение регистра eax (выходной операнд должен указываться прежде входных и потому виден как %0). Регистр eax помечен в ассемблерной вставке как разрушаемый, а потому нет необходимости сохранять и восстанавливать его значение через стек – эта задача передана компилятору GCC.

Задание для выполнения

1. Разобраться с запуском команд по расписанию. Научиться выполнять по расписанию вывод текстового сообщения в консоль, а также запуск текстового редактора.

2. Написать модуль ядра, осуществляющий запись данных в аппаратный буфер контроллера клавиатуры (результат записи символа в буфер клавиатуры равнозначен нажатию соответствующей клавиши пользователем). Модуль должен производить это действие при своей загрузке в память. Таким образом периодической загрузкой модуля можно имитировать наличие пользователя за компьютером. Для передачи данных в буфер клавиатуры использовать ассемблерную вставку с параметрами.

3. Настроить cron таким образом, чтобы он запускал через заданный интервал времени загрузку и выгрузку созданного модуля ядра. Обращение к командам insmod и rmmod выполнять через команду sudo.

ЛАБОРАТОРНАЯ РАБОТА №5-7
«ИЗУЧЕНИЕ ОБРАБОТЧИКОВ СИСТЕМНЫХ ВЫЗОВОВ. ЗАМЕНА ОБРАБОТЧИКА СИСТЕМНОГО ВЫЗОВА ВО ВРЕМЯ КОМПИЛЯЦИИ И ВО ВРЕМЯ РАБОТЫ МОДУЛЯ»

1 Краткие теоретические сведения

Данная лабораторная работа предназначена для запуска на операционной системе Ubuntu версии 14. Эта версия обеспечивает оптимальную работу программ и инструментов, необходимых для выполнения лабораторной работы. При использовании других версий Ubuntu или других операционных систем могут возникнуть проблемы совместимости или ошибки. Поэтому рекомендуется установить Ubuntu 14 на свой компьютер или использовать виртуальную машину для запуска лабораторной работы. Адреса обработчиков системных вызовов хранятся ядром в таблице системных вызовов (sys_call_table). Обработчик, расположенный по одному из этих адресов, вызывается каждый раз, когда какая-то программа вызывает прерывание 80h с номером какого-то системного вызова в регистре eax (например, eax=4 для системного вызова write, выполняющего запись в файл или устройство вывода).

Зная адрес этой таблицы и номер нужного вызова, можно подменить его обработчик своим собственным кодом.

Примечание: чтобы полностью заменить системный вызов своим кодом, нужно полностью реализовать его функционал. Поэтому чаще поступают по-другому: при подмене адреса в таблице системных вызовов сохраняют прежнее значение в какой-либо переменной, и каждый раз после выполнения своих действий передают управление на этот адрес. Такой подход позволяет добавить собственные действия без ущерба для уже существующего функционала и таким образом не сломать работу операционной системы.

2 Подмена системных вызовов

Таким образом, можно написать модуль ядра, реализовать в нем функцию с кодом нового обработчика, и подменить стандартный обработчик адресом этой функции.

Главная задача, которую требуется решить для подмены обработчика — выяснить расположение таблицы системных вызовов в оперативной памяти. Рассмотрим два различных подхода к решению этой задачи: поиск адреса во время компиляции и поиск адреса во время работы программы (когда модуль ядра уже загружен в память).

2.1 Поиск адреса таблицы вызовов во время компиляции

Идея данного метода заключается в выполнении команд оболочки (bash) в Makefile. Адрес нужного системного вызова можно найти в файле “System.map-версия_ядра”. Найденный адрес можно добавить в компилируемый модуль.

Ниже приведена команда для поиска адреса системного вызова в файле “System.map-версия_ядра”:

```
grep sys_call_table /boot/System.map-$(uname -r) | awk '{print $1}'
```

Команда выведет на экран найденный адрес, например:

```
c05d3180
```

Примечание: на системах, поддерживающих более одной архитектуры, команда может выдать больше одного адреса. Типичным примером является Linux для 64-битной архитектуры, т. к. его ядро содержит две таблицы: sys_call_table для «родных» 64-битных системных вызовов, и ia32_sys_call_table для 32-битных системных вызовов, присутствующих для совместимости с 32-битными программами. Как и остальные работы данного курса, данная работа

касается 32-битных системных вызовов; поэтому, если задание выполняется на 64-битной системе, вместо sys_call_table аргументом команды grep должна быть подстрока ia32_sys_call_table.

Чтобы не подставлять найденный адрес вручную, можно написать исполняемый скрипт (например, назвав его compile.sh), который выполняет следующие действия:

– ищет адрес в таблице

системных вызовов; – вставляет

его в С-программу; – запускает

Makefile.

Ниже приведен текст такого файла (“compile.sh”):

```
#!/bin/bash

TABLE=$(grep ia32_sys_call_table /boot/System.map-$(uname -r) | awk '{print
$1}')
sed -i s/TABLE/$TABLE/g hijack.c
make -C /usr/src/linux-headers-`uname -r` SUBDIRS=$PWD modules
```

Файл скрипта должен быть исполняемым (иметь право на запуск), а в одном с ним каталоге должны находиться программа (в примере это файл hijack.c), а также Makefile для сборки модуля.

Пример подмены обработчика

В примере ниже использован командный текстовый редактор sed для модификации исходного файла программы. Командный файл должен быть исполняемым. Модификация исходного кода заключается в замере слова TABLE найденным адресом.

Текст файла “Makefile”:

```
obj-m   := hijack.o
```

Текст модуля ядра (“hijack.c”), в исходном коде которого будет будет заменяться адрес представлен ниже.

Как видно, этот код подменяет обработчик системного вызова write своей функцией new_write().

Адрес прежнего обработчика сохраняется в переменной (указателе на функцию), которая имеет имя original_write.

Новый обработчик при вызове выводит в лог-файл ядра сообщение WRITE HIJACKED, а затем передает управление прежнему обработчику. Подмена

обработчика выполняется в функции init() при загрузке модуля в память, а при выгрузке функция exit() должна восстановить прежнее значение обработчика (иначе, выгрузив модуль, система сразу умрет при попытке что-то вывести).

Напомним, что указатель на функцию — это такая специальная разновидность указателей, которую используют для вызова функций, находящихся по неизвестному заранее адресу. При объявлении переменной – указателя на функцию необходимо сообщить транслятору типы возвращаемого значения и аргументов для вызываемой функции (чтобы транслятор знал, чего и сколько класть в стек). Например, объявление указателя для системного вызова write() выглядит так: «asmlinkage int (*ukazatel_na_write)(unsigned int, const char __user *, size_t)».

Текст модуля ядра (“hijack.c”):

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>
#include <linux/types.h>
#include <linux/unistd.h>
#include <asm/cacheflush.h>
#include <asm/page.h>
#include <asm/current.h>
#include <linux/sched.h>
#include <linux/kallsyms.h>

unsigned long *syscall_table = (unsigned long *)0xTABLE;
asmlinkage int (*original_write)(unsigned int, const char __user *, size_t);

asmlinkage int new_write(unsigned int fd, const char __user *buf, size_t count)
{
    // измененная функция write
printk(KERN_ALERT "WRITE HIJACKED");     return
(*original_write)(fd, buf, count);
}
static int init(void) {
    printk(KERN_ALERT "\nHIJACK INIT\n");
write_cr0 (read_cr0 () & (~ 0x10000));
    original_write = (void
*)syscall_table[__NR_write];
syscall_table[__NR_write] = new_write;       write_cr0
(read_cr0 () | 0x10000);     return 0;
}
static void exit(void) {
    write_cr0 (read_cr0 () & (~ 0x10000));
```

```
syscall_table[__NR_write] = original_write;
write_cr0 (read_cr0 () | 0x10000);
printk(KERN_ALERT "MODULE EXIT\n");     return;
}
module_init(init);
module_exit(exit);
```

Идентификатор asmlinkage уведомляет компилятор gcc, что функция будет искать свои аргументы не в регистрах, а в стеке. Это необходимо, т. к. современные компиляторы при оптимизации кода норовят передавать аргументы функциям через незанятые регистры, экономя таким образом на обращениях к памяти. Для системных вызовов эту оптимизацию нужно отключать.

Обход защиты памяти

При подмене обработчика необходимо обойти защиту от записи для области векторов прерываний. Мы делаем это сбросом WP-бита системного регистра CR0. Этот бит действует на аппаратном уровне, разрешая (для кода, имеющего достаточные привилегии) модификации страниц памяти независимо от того, разрешена ли в них запись или нет. Доступ к регистру CR0 выполняется макросами write_cr0() и read_cr0().

Примечание: эти макросы устроены предельно просто; write_cr0(x) просто выполняет инструкцию «mov cr0, x».

Сборка и запуск примера

Можно скомпилировать код модуля, получив на экране нечто в таком роде:

```
user:~$ sh compile.sh
make -C /lib/modules/2.6.35-24-generic/build
SUBDIRS=/home/user/Hacking/Syscall_Hijack/mod2 modules
make[1]: entering directory "/usr/src/linux-headers-2.6.35-24-generic"
  CC [M]  /home/user/Hacking/Syscall_Hijack/mod2/hijack.o
/home/user/Hacking/Syscall_Hijack/mod2/hijack.c: In function ‘init’:
/home/user/Hacking/Syscall_Hijack/mod2/hijack.c:33: warning: assignment makes
integer from pointer without a cast
/home/user/Hacking/Syscall_Hijack/mod2/hijack.c: In function ‘exit’:
/home/user/Hacking/Syscall_Hijack/mod2/hijack.c:44: warning: assignment makes
integer from pointer without a cast
  Building modules, stage 2.
  MODPOST 1 modules
  LD [M]  /home/user/Hacking/Syscall_Hijack/mod2/hijack.ko
make[1]: leaving directory "/usr/src/linux-headers-2.6.35-24-generic"
```

После сборки модуля, если открыть файл “hijack.c”, можно увидеть вместо “TABLE” адрес таблицы системных вызовов. Например:

```
user:~$ cat hijack.c |grep *syscall_table
unsigned long *syscall_table = (unsigned long *)0xc05d3180; user:~$
```

Можно запустить модуль:

```
user:~$ sudo insmod hijack.ko
user:~$
```

Результат работы модуля должен быть виден в вызове команды dmesg.

Это очень простой и незамысловатый способ, но есть более удобный и гибкий.

2.2 Поиск адреса таблицы во время работы модуля

Недостаток приведенного выше способа — созданный модуль нужно перекомпилировать на каждой новой системе, из-за того что адрес таблицы системных вызовов на ней может быть другим. Чтобы обойти это ограничение, необходимо находить этот адрес в момент выполнения модуля, а не в момент его компиляции.

Для получения адреса таблицы системных вызовов во время работы модуля можно воспользоваться следующей последовательностью действий:

1) Найти, где система хранит версию используемого ядра. Это нужно, чтобы на следующем этапе открыть файл с именем “/boot/System.map-версия_ядра” и отыскать в нем адрес таблицы.

Примечание 1. если вы выполняете данную лабораторную работу в аудитории 306 во втором корпусе, обратите внимание на то, что функция open работает только на двух старых компьютерах, расположенных в конце аудитории. На остальных она может вызывать ошибки или зависания. В случае возникновения каких-либо проблем с ними обратитесь к преподавателю или лаборанту.

Примечание 2. для того, чтобы в программе прочитать версию используемого ядра, можно открыть файл /proc/version и прочитать всё его содержимое. Учитывая, что мы находимся в ядре, лучше использовать для открытия файла функцию filp_open:

```
struct file *f;
static char *fname = "/proc/version";  static
char *fMapName, *str2, *str3, *str4;  static
char *str1 = "/boot/System.map-";  static
char buff[501];  size_t n, i, sizeBuff = 0;
```

```
// Открытие файла /proc/version   filp_open(fname,
O_RDONLY, 0);
// Извлечение данных из файла
n = kernel_read(f, 0, buff, 500);
if(n) {   buff[n] = 0;
}
filp_close(f, current->files);
```

В результате в переменной buff будет храниться строка вида:

```
     Linux version 2.6.35-24-generic (buildd@vernadsky) (gcc version 4.4.5
(Ubuntu/Linaro 4.4.4-14ubuntu5) ) #42-Ubuntu SMP Thu Dec 2 01:41:57 UTC 2010
```

Заметим, что такую же строку вы можете увидеть в терминале, выполнив:

```
cat /proc/version
```

Из всей этой строки мы хотим получить только версию используемого ядра (в нашем примере это «2.6.35-24-generic»)

Примечание 3. Кроме версии ядра, файл содержит некоторую дополнительную информацию (такую как использованный для сборки ядра компилятор), поэтому содержимое прочитанной из файла строки нужно обрезать, чтобы получить только последовательность цифр с версией. Полное содержимое файла /proc/version может выглядеть, например, так:

```
      Linux version 2.6.35-24-generic (buildd@vernadsky) (gcc version 4.4.5
(Ubuntu/Linaro 4.4.4-14ubuntu5) ) #42-Ubuntu SMP Thu Dec 2 01:41:57 UTC
2010
```

2) Выделив из содержимого номер версии ядра (в приведенном примере это подстрока «2.6.35-24-generic»), следует добавить его в конец последовательности символов “/boot/System.map-", а затем открыть файл с получившимся именем.

Примечание: можно получить ту же строку с номером ядра, воспользовавшись системным вызовом uname. Подробнее узнать об это можно, выполнив «man 2 uname».

3) В открытом файле “/boot/System.map-версия_ядра” и выполняется собственно поиск адреса таблицы системных вызовов. Чтение файла выполняется строка за строкой, пока не будет найдена подстрока “sys_call_table”.

4) Далее необходимо выполнить преобразование полученного адреса в шестнадцатеричную форму.

Примечание: в процессе выполнения лабораторной работы вы можете столкнуться с ситуацией, когда имя файла не выводится при открытии функции open. Это может быть связано с тем, что функция не работает корректно или не подходит для данного типа файла. В таком случае следует удалить функцию из вашего кода. Если проблема повторяется, обратитесь к преподавателю или лаборанту.

3. ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ

1) Ознакомиться с понятием подмены системного вызова, а также способами, которыми можно осуществить это действие.

2) Написать модуль ядра, подменяющий обработчик open и выводящий в лог ядра диагностическое сообщение: такое же, как в примере для системного вызова write, но содержащее в себе имя открываемого файла. Проверить работоспособность созданного модуля.

3) Модифицировать модуль ядра так, чтобы адрес таблицы системных вызовов определялся в процессе загрузки модуля в память.

4) Модифицировать функцию, являющуюся новым обработчиком системного вызова, написав ее на ассемблере (оформить ее либо как ассемблерную вставку, либо как отдельный ассемблерный файл, содержащий в себе процедуру).

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем заключается суть подмены системного вызова? Какова может быть цель такого действия?
2. Как использование таблицы системных вызовов связано с механизмом обработки прерываний?
3. В какой момент должен срабатывать обработчик, подменяющий собой системный вызов?
4. Для чего обработчик, подменяющий собой системный вызов, должен также вызвать прежний обработчик?
5. Объясните, что делает код, выполняющий поиск таблицы системных вызовов при компиляции программы.

ЛАБОРАТОРНАЯ РАБОТА №8
«РАБОТА СО ЗВУКОВЫМИ УСТРОЙСТВАМИ»

Цель работы: научиться работать со звуковой подсистемой GNU/Linux, включая вывод аудиофайлов, получение информации об аппаратных возможностях аудио-карты и регулирование громкости отдельных каналов микшера.

Краткие теоретические сведения

1. Звуковая подсистема GNU/Linux

В Linux, как и в других UNIX-совместимых системах, пользовательские программы редко обращаются к аппаратному обеспечению напрямую. У ядра есть драйверы для всех устройств. Все вопросы, касающиеся отличий звуковых карт, берет на себя модуль ядра, отвечающий за поддержку конкретной аудиокарты — то есть *драйвер* этого конкретного аудио-устройства.

В соответствии с традициями Unix, в виртуальной файловой системе /dev/ присутствуют несколько стандартных файлов устройств, обращаясь к которым можно менять настройки аудио-карты (например, уровень громкости на различных каналах), выводить звук на линейный выход или считывать звук с линейного входа. Благодаря драйверу аудио-карты, взаимодействие с этими файлами устройств не зависит от того, какая именно аудио-карта установлена в компьютере, и можно легко создавать универсальные программы, одинаково работающие со звуком на разных системах. Знание конкретных адресов портов и команд микроконтроллера, отвечающего за ввод, обработку и вывод звука (т. н, аудио-процессор, от англ. digital sound processor или DSP) требуется только разработчику аудио-драйвера (и это очень хорошо, поскольку аудиокарты слабо стандартизированы).

Файловая абстракция очень удобна для работы со звуком: например, запись звукового файла байт за байтом в специальный файл устройства приводит к тому, что файл проигрывается на подключенных к компьютеру колонках (или наушниках).

В GNU/Linux на сегодняшний день существует несколько аудиоподсистем, файлы устройств у которых несколько различаются:

OSS (от англ. Open Sound System) - исторически первая аудио-подсистема, использовавшаяся раньше в Linux и до сих пор активно используемая во многих Unix-подобных системах;

Alsa (сокращение от Andvanced Linux Sound Architecture) — более новая аудио-подсистема, разработанная специально для Linux, и изначально поддерживающая ряд дополнительных возможностей более дорогих аудио-карт, таких как аппаратное смешивание (*микширование*) звуковых каналов, полнодуплексный режим и т. д.

PulseAudio — высокоуровневая надстройка над аудио-подсистемой (обычно над Alsa), умеющая выполнять программное микширование звука, трансляцию звука по сети, эмуляцию других аудио-подсистем и т. д. Дальше мы будем рассматривать файлы устройств для работы со звуком, существующие в звуковой подсистеме OSS. Остальные системы умеют эмулировать работу OSS, а потому этот подход наиболее универсален.

Наиболее часто используемые файлы устройств в системе OSS:

/dev/mixer — позволяет настроить уровни громкости для различных входных и выходных каналов звуковой карты; используется для доступа к аппаратному микшеру, встроенному в звуковую карту (по этой причине возможности, поддерживаемые /dev/mixer, различаются для разных аудиокарт).

/dev/dsp — используется для вывода звука (когда в файл устройства записываются несжатые аудио-данные) или для считывание звука с аудиовходов (когда выполняется чтение из файла устройства).

На системах, в которых используется звуковая система PulseAudio (например, все современные версии Ubuntu Linux) файлы устройств в стиле OSS могут отсутствовать. Для их эмуляции используется специальная утилита padsp. Программа, работающая со звуком через /dev/dsp, будет «видеть» необходимые файлы устройств, если будет запущена этой утилитой. Например, запуск программы myMediaPlayer будет выглядеть так:

```
padsp myMedaPlayer
```

Этот способ запуска мы будем использовать далее при выполнении заданий лабораторной работы.

1.1. Воспроизведение звука командой cat

Простейший способ воспроизвести звук — перенаправить звуковой файл в устройство /dev/dsp с помощью команды cat:

```
cat myNewSound >/dev/dsp
```

Файл myNewSound в приведенном примере должен быть в несжатом аудио-формате. Например, для этой цели подходит обычный WAV-файл (формат, известный также как Windows PCM).

Соответственно, простейший способ записи звука выглядит точно так же, но данные не записываются из файла в /dev/dsp, а наоборот — читаются из /dev/dsp в файл:

```
cat /dev/dsp1 >myNewSound
```

1.2. Программирование аудио-устройств

Файл устройства /dev/mixer

Каналы микшера можно разделить на 2 группы: входные (для записи звука) и выходные (для его проигрывания). Из-за большого числа и разнообразия каналов для регулирования уровней звука с помощью /dev/mixer применен более сложный подход, чем простые файловые операции чтения и записи. Стандартные операции, необходимые для работы с микшером — это открытие файла (системный вызов open), его закрытие (системный вызов close), а также не использовавшийся нами ранее системый вызов ioctl, через который и выполняются все настройки.

Системный вызов ioctl

Прототип функции ioctl выглядит следующим образом:

```
int ioctl(int fd, int request, ...);
```

Функция используется для выполнения таких операций над файлами и устройствами, которые не входят в число операций чтения и записи. Каждый запрос специфичен для различных устройств.

Первый параметр функции — это файловый описатель (тот самый, который вернула функция open). Второй — тип нужной нестандартной операции. Необязательный третий параметр специфичен для конкретных устройств.

Параметры микшера

Константы, соответствующие наиболее типичным каналам микшера, приведены в следующей таблице. Обратите внимание, что наряду с реальными каналами звукового тракта (линейный вход, линейный выход, вход для подключения audio-CD) в микшере присутствуют «виртуальные каналы», используемые для регулировки каких-либо параметров звукового тракта (например, аудиоустройство может иметь «канал» для регулировки высоких или низких частот).

Таблица 1 — Константы, обозначающие названия каналов микшера

Название канала	**Описание**
SOUND_MIXER_VOLUME	Канал одновременной регулировки громкости для всего сразу
SOUND_MIXER_BASS	Канал для регулировки низких частот
SOUND_MIXER_TREBLE	Кнал для регулировки высоких частот
SOUND_MIXER_SYNTH	FM-синтезатор
SOUND_MIXER_PCM	Уровень цифро-аналогового преобразователя (ЦАП)
SOUND_MIXER_SPEAKER	Громкость системного динамика
SOUND_MIXER_LINE	Линейный вход
SOUND_MIXER_MIC	Вход для подключения микрофона
SOUND_MIXER_CD	Вход audio-CD
SOUND_MIXER_IMIX	Громкость воспроизведения с устройства записи
SOUND_MIXER_ALTPCM	Второй ЦАП
SOUND_MIXER_RECLEV	Регулировка громкости при записи звука, действующая сразу на все входные каналы
SOUND_MIXER_IGAIN	Входной уровень усиления звука
SOUND_MIXER_OGAIN	Выходной уровень усиления звука

Приведем пример определения уровня громкости на входе для подключения микрофона:

```
int vol;
ioctl(fd, MIXER_READ(SOUND_MIXER_MIC), &vol);
printf("Уровень усиления звука на входе: %d %%\n", vol);
```

В этом примере предполагается, что файл /dev/mixer был предварительно открыт функцией open(), и эта функция вернула описатель файла в переменную fd.

Теперь поясним, что представляет собой указатель vol (третий параметр функции). При использовании для чтения параметров микшера, функция ioctl() сохраняет по этому адресу считанное значение громкости.

Такой запутанный способ получения значения (вместо возвращаемого значения функции) используется из-за того, что многие каналы микшера стереофонические. Для стерео-каналов громкость включает два значения (по одному на левый и правый канал). Если по адресу vol была сохранена пара значений, то первый байт соответствует правому каналу, а последний — левому:

```
int left, right;
left  =  vol & 0xff;
right = (vol & 0xff00) >> 8;
printf("Громкость левого канала %d %%, правого канала %d %%\n", left, right);
```

Для моно-устройств (один канал) уровень шума находится в младшем байте.

Установка уровней громкости производится аналогично:

```
vol = (right << 8) + left;
ioctl(fd, MIXER_WRITE(SOUND_MIXER_MIC), &vol);
```

Дополнительно можно получить символьные имена каналов:

```
const char *labels[] = SOUND_DEVICE_LABELS;
const char *names[]  = SOUND_DEVICE_NAMES;
```

Для поиска информации о микшере используется несколько вызовов ioctl. Все они возвращают битовые маски, соответствующие каналам микшера. SOUND_MIXER_READ_DEVMASK возвращает битовую маску, где для каждого, поддерживаемого микшером канала установлен соответствующий бит. SOUND_MIXER_READ_RECMASK — биты для каждого канала, который может быть использован, как устройство записи. Для примера, можно проверить, поддерживается ли микшером канал CD-входа:

```
ioctl(fd, SOUND_MIXER_READ_DEVMASK, &devmask);
if (devmask & SOUND_MIXER_CD)
 printf("CD-вход поддерживается");
```

Так же можно определить, доступен ли этот канал как устройство записи:

```
ioctl(fd, SOUND_MIXER_READ_RECMASK, &recmask);
```

```
if (recmask & SOUND_MIXER_CD)
  printf("CD-вход можно использовать для записи звука");
```

SOUND_MIXER_READ_RECSRC — каналы, выбранные сейчас для записи.

SOUND_MIXER_READ_STEREODEVS — каналы, поддерживающие стерео.

Похожий вызов ioctl возвращает информацию о звуковой плате в целом: SOUND_MIXER_READ_CAPS. Каждый бит соответствует возможностям карты. Сейчас существует один бит: SOUND_CAP_EXCL_INPUT. Если он установлен — источников записи может быть несколько.

SOUND_MIXER_WRITE_RECSRC — источник записи.

Пример, устанавливающий CD в качестве источника записи:

```
devmask = SOUND_MIXER_CD;
ioctl(fd, SOUND_MIXER_WRITE_DEVMASK, &devmask);
```

Обратите внимание, что функции необходимо передавать указатель.

Пример

Пример ниже иллюстрирует некоторые концепции системных вызовов:

```
/*
 * syscalls.c
 * Программа только иллюстрирует работу с системными вызовами
 * пока не делает ничего полезного но далее будет расширена
 */
#include <unistd.h>
#include <stdio.h> #include <fcntl.h>
#include <sys/types.h>
#include <sys/ioctl.h> #include <linux/soundcard.h>

int main()
{
    int fd                    /* описатель файла */
    int arg;                  /* аргумент к вызову ioctl */
    unsigned char buf[1000]; /* буфер для данных */
    int status;              /* возвращаемый статус */

/* открытие устройства */
status = fd = open("/dev/dsp", O_RDWR);
```

```
if (status == -1) {
    perror("error opening /dev/dsp");
    exit(1);
}

/* задание параметров     ioctl */
arg = 8000; /* частота дискретизации */
status = ioctl(fd, SOUND_PCM_WRITE_RATE, &arg);
if (status == -1) {
    perror("error from SOUND_PCM_WRITE_RATE ioctl");
    exit(1);
}

/* чтение */
status = read(fd, buf, sizeof(buf));
if (status == -1) {
    perror("error reading from /dev/dsp");
    exit(1);
}

/* запись */
status = write(fd, buf, sizeof(buf));
if (status == -1) {
    perror("error writing to /dev/dsp");
    exit(1);
}

/* закрытие */
status = close(fd);
if (status == -1) {
    perror("error closing /dev/dsp");
    exit(1);
}

/*  и выход */
return(0);

}
```

1.3 Программирование /dev/dsp

/dev/dsp — это цифровое устройство воспроизведения и записи. Запись в устройство приводит к воспроизведению звука (ЦАП). Чтение — запись звука и его анализ (АЦП).

Если данные читать слишком медленно, то переполненный буфер приведет к «отбрасыванию» лишних данных, из-за чего звук станет воспроизводиться рывками. Если читать слишком быстро, то ядро будет блокировать запросы до тех пор, пока не накопится необходимое количество данных. Само собой, в конце воспроизведения или записи не будет сообщения о конце файла.

Формат принимаемых данных зависит от того, как устройство было настроено вызовом ioctl. Стандартные настройки таковы: 8-битные беззнаковые дискретные отсчеты аудио-потока (сэмплы) с одним каналом (моно) и частотой дискретизации 8кГц. Аналогичные настройки по умолчанию приняты и для записи.

Константы ioctl описаны в заголовочном файле: *linux/soundcard.h*.

SOUND_PCM_WRITE_BITS

SOUND_PCM_READ_BITS

SOUND_PCM_WRITE_CHANNELS

SOUND_PCM_READ_CHANNELS

SOUND_PCM_WRITE_RATE

SOUND_PCM_READ_RATE

Пример

Ниже приводится короткий пример программирования DSP, записывающий несколько секунд звука в массив в памяти и затем воспроизводящий его.

```
/*
* выходит по ctrl-c
*/
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <stdlib.h>
#include <stdio.h>
#include <linux/soundcard.h>
```

```
#define LENGTH 3    /* сколько секунд записывать */
#define RATE 8000   /* частота дискретизации */
#define SIZE 8      /* глубина: 8 или 16  */
#define CHANNELS 1  /* 1 = моно 2 = стерео */

/* буфер для цифрового звука */
unsigned char buf[LENGTH*RATE*SIZE*CHANNELS/8];

int main()
{
    int fd;       /* файловый описатель */
    int arg;      /* аргумент к  ioctl */
    int status;   /* возвращаемый системой результат */

/* открытие устройства */
    fd = open("/dev/dsp", O_RDWR);
    if (fd < 0) {
     perror("open of /dev/dsp failed");
     exit(1);
}

/* задание параметров */
arg = SIZE;      /* размер сэмпла */
status = ioctl(fd, SOUND_PCM_WRITE_BITS, &arg);
if (status == -1)
    perror("SOUND_PCM_WRITE_BITS ioctl failed");
if (arg != SIZE)
    perror("unable to set sample size");

arg = CHANNELS;  /* моно или стерео */
status = ioctl(fd, SOUND_PCM_WRITE_CHANNELS, &arg);
if (status == -1)
    perror("SOUND_PCM_WRITE_CHANNELS ioctl failed");
if (arg != CHANNELS)
    perror("unable to set number of channels");
arg = RATE;      /* частота дискретизации */
status = ioctl(fd, SOUND_PCM_WRITE_RATE, &arg);
if (status == -1)
    perror("SOUND_PCM_WRITE_WRITE ioctl failed");
```

```
while (1) { /* loop until Control-C */
    printf("Say something:\n");
    status = read(fd, buf, sizeof(buf)); /* запись звука */
    if (status != sizeof(buf))
     perror("read wrong number of bytes");
    printf("You said:\n");
    status = write(fd, buf, sizeof(buf)); /* воспроизведение */
    if (status != sizeof(buf))
     perror("wrote wrong number of bytes");
    /* wait for playback to complete before recording again */
    status = ioctl(fd, SOUND_PCM_SYNC, 0);
    if (status == -1)
     perror("SOUND_PCM_SYNC ioctl failed");
     }
}
```

Задания для выполнения:

1. Внести незначительные изменения в программу для работы с /dev/dsp: частота дискретизации, размер, время записи. Описать изменение качества звука при изменении частоты дискретизации.
2. Воспроизвести звуковой сэмпл в обратном порядке (для /dev/dsp).
3. Написать программу, позволяющую выбирать устройство записи в микшере (/dev/mixer).
4. Получить данные о возможностях аудио-карты, воспользовавшись для этого следующими константами:
   - SOUND_MIXER_READ_RECSRC,
   - SOUND_MIXER_READ_DEVMASK,
   - SOUND_MIXER_READ_RECMASK,
   - SOUND_MIXER_READ_STEREODEVS, - SOUND_MIXER_READ_CAPS.

   Вывести информацию о возможностях аудио-карты в виде следующей таблицы:

```
Status of /dev/mixer:
Mixer      Device     Recording  Active     Stereo     Current
Channel    Available  Source     Source     Device     Level
---------  ---------  ---------  --------   ---------  ---------
 0 Vol      yes         no         no         yes        90% 90%
 1 Bass     no          no         no         no
 2 Trebl    no          no         no         no
 3 Synth    yes         no         no         yes        75% 75%
 4 Pcm      yes         no         no         yes        75% 75%
 5 Spkr     no          no         no         no
 6 Line     yes         yes        no         yes        75% 75%
 7 Mic      yes         yes        yes        no         16%
 8 CD       yes         yes        no         yes        75% 75%
 9 Mix      no          no         no         no
10 Pcm2     no          no         no         no
11 Rec      no          no         no         no
12 IGain    no          no         no         no
13 OGain    no          no         no         no
14 Line1    no          no         no         no
15 Line2    no          no         no         no
16 Line3    no          no         no         no
```

3. РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ

3.1. ПЕРЕЧЕНЬ ВОПРОСОВ К ЭКЗАМЕНУ

1. Особенности архитектуры фон Неймана.
2. Особенности гарвардской архитектуры.
3. Состав и схема взаимодействия блоков ПК.
4. Что такое микропроцессор?
5. Понятия архитектуры, микроархитектуры и макроархитектуры микропроцессора. Типы архитектуры процессора
6. Характеристики микропроцессора.
7. Что определяет технологический процесс производства микропроцессоров?
8. Функции микропроцессора.
9. Что такое шина? Какие виды шин используются в микропроцессорной схеме?
10. Что понимают под «разрядностью ЭВМ»?
11. Внутренняя организация микропроцессора (упрощенная схема).
12. Структура и состав современного процессора.
13. Способы повышения производительности ядра процессора.
14. Что такое конвейеризация процессора?
15. Что такое суперскалярность процессора?
16. Что понимают под чипсетом материнской платы?
17. Назначение южного моста.
18. Назначение северного моста.
19. Материнские платы. Структура. Виды форм-факторов.
20. Что такое BIOS?
21. Функции BIOS.
22. Тест начального включения – POST.
23. Звуковая диагностика POST. Диагностические сообщения POST.
24. Процедура начальной загрузки Bootsrap loader.
25. Утилита Setup BIOS’a.
26. Что такое область данных BIOS’a?
27. UEFI.
28. Какие устройства называют системными?
29. Назначение тактового генератора

30. Назначение и состав часов реального времени
31. Системный таймер: назначение, состав, режимы работы.
32. Каналы системного таймера.
33. Устройства для отсчета времени в ПК: системный таймер, Local APIC, HPET, TSC
34. Стандарт ACPI
35. Что содержит таблица векторов прерываний?
36. Типы прерываний, их источники и приоритеты, коды векторов.
37. Процедура обработки прерывания.
38. Функции контроллера прерываний.
39. Назначение, состав и работа контроллера прерываний.
40. Каскадирование контроллера прерываний.
41. Улучшенный контроллер прерываний APIC.
42. Что такое DMA?
43. Контроллер DMA.
44. Что такое прямое управление шиной?
45. Что такое регенерация памяти?
46. Современная реализация механизма ПДП.
47. Что такое файл?
48. Что такое файловая система?
49. Что такое система управления файлами?
50. Разбиение дисков на разделы (MBR-разметка).
51. Назначение таблицы разделов жесткого диска (MBR-разметка).
52. GPT-разметка диска
53. Файловая система FAT: назначение и состав,
54. Файловая система NTFS: характеристики и структура.
55. Организация и адресация информации на магнитном диске: chs и LBA.
56. Конструкция НЖМД: гермозона и плата электроники, их состав и функционирование.
57. Параметры винчестеров.
58. Основы магнитной записи
59. Что такое последовательный интерфейс? примеры
60. Что такое параллельный интерфейс? примеры
61. Интерфейсы НЖМД.
62. Иерархия памяти.

63. КЭШ-память: назначение, расположение, типовая структура.
64. Логическая организация кэш-памяти процессора
65. Функции контроллера памяти.
66. Функции системы управления памятью.
67. Концепция виртуальной памяти.
68. Страничная организация памяти.
69. Что содержится в таблице страниц?
70. Сегментная организация памяти.
71. Схемы управления памятью.
72. Что такое трансляция адреса?
73. Суть механизма страничного обмена.
74. Назначение файла подкачки.
75. Видеосистема: назначение, состав и параметры.
76. Назначение графического контроллера.
77. Назначение графического процессора.
78. Что такое видеопамять?
79. Организация видеопамяти в графическом и текстовом режиме.
80. Принцип работы ЭЛТ-монитора.
81. ЖК-монитор: устройство, характеристики.
82. Что такое жидкий кристалл?
83. Что такое устройства ввода. Классификация. Примеры.
84. Что такое устройства вывода. Примеры.
85. Кодирование текстовой информации.
86. Кодирование графической информации.
87. Клавиатура: принцип работы, контроллеры клавиатуры, порты и команды.
88. Что такое скан-код? Что такое ASCII-код?
89. Принтеры. Технологии печати.

4. ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

4.1 УЧЕБНАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Учреждение образования
«Брестский государственный технический университет»

УТВЕРЖДАЮ

Первый проректор

_____ М.В.Нерода

« »_____20 г.

Регистрационный № УД-_____
/уч.

Архитектура персональных компьютеров

Учебная программа учреждения высшего образования по учебной дисциплине для специальности

1-40 02 01 Вычислительные машины, системы и сети

Брест 2022

Учебная программа составлена на основе типовой учебной программы дисциплины «Архитектура персональных компьютеров» для специальности 1-40 02 01 Вычислительные машины, системы и сети, рег. № ТД-I.1563/тип от 05.05.2022.

СОСТАВИТЕЛИ:

М.В. Николаюк-Ртищева, старший преподаватель кафедры электронных вычислительных машины и систем

Г.А. Четверкина, старший преподаватель кафедры электронных вычислительных машины и систем

РЕКОМЕНДОВАНА К УТВЕРЖДЕНИЮ:

Кафедрой электронных вычислительных машин и систем

(протокол от _____ № ____);

Заведующий кафедрой С.С.Дереченник

Методической комиссией факультета электронно-информационных систем

(протокол от _____ № ____);

Председатель комиссии С.С.Дереченник

Научно-методическим советом БрГТУ (протокол № ___ от _____)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Учебная дисциплина «Архитектура персональных компьютеров» относится к государственному компоненту и входит в модуль «Архитектура компьютеров» учебного плана специальности.

Целью изучения дисциплины «Архитектура персональных компьютеров» является изучение организации и функционирования аппаратуры персональных ЭВМ, а также приобретение знаний и навыков в области программирования устройств современных компьютеров (базовая профессиональная компетенция БПК-14)

Задачи учебной дисциплины – приобретение знаний о строении, принципах функционирования и взаимодействия компонентов архитектуры современных персональных компьютеров, а также формирование навыков настройки, администрирования и программирования компонентов архитектуры современных персональных ЭВМ.

В результате освоения дисциплины студенты должны:

знать:

- архитектуру компьютеров, принципы функционирования и взаимодействия компонентов материнской платы, периферийных устройств;
- принципы организации различных архитектур ПК;
- характеристики, состав и принципы работы основных устройств ПК;
- способы взаимодействия с устройствами ПК;
- принципы работы основных периферийных устройств и способы взаимодействия с ними;

уметь:

- оценивать характеристики системных и периферийных устройств ПК;
- писать программные приложения взаимодействия с системными и периферийными устройствами;

иметь представление о сборке ПК и настройке его оборудования.

Дисциплина базируется на знаниях, приобретенных обучающимися при изучении учебной дисциплины «Арифметические и логические основы цифровых устройств», «Основы алгоритмизации и программирования». Материал дисциплины ««Архитектура персональных компьютеров»

используется при изучении ряда последующих дисциплин учебного плана, входящих в модули «Архитектура компьютеров», «Информационно-управляющие системы», «Мультиплатформенные системы».

Дисциплина преподается в третьем семестре обучения. Программа рассчитана на 180 учебных часов. Количество аудиторных часов, а также их распределение по видам занятий указано в тематическом плане. Форма контроля знаний – экзамен.

План учебной дисциплины для дневной формы получения высшего образования

Код специальности (направления специальности)	Наименование специальности (направления специальности)	Курс	Семестр	Всего учебных часов	Количество зачетных единиц	Аудиторных часов (в соответствии с учебным планом УВО)						Академических часов на курсовой проект (работу)	Форма текущей аттестации
						Всего	Лекции	Лабораторные занятия	Практические занятия	Семинары	Управляемая самостоятельная работа		
1-40 02 01	Вычислительные машины, системы и сети	2	3	180	5.0	80	48	32	–	–	–	–	экзамен

СОДЕРЖАНИЕ УЧЕБНОГО МАТЕРИАЛА

Лекционный курс

ВВЕДЕНИЕ

Предмет курса, его цели и задачи. История развития вычислительных систем. Понятие архитектуры персонального компьютера. Гарвардская и принстонская архитектура.

Раздел 1. КЛАССИФИКАЦИЯ АРХИТЕКТУР ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Общие сведения о ПЭВМ. Классификация Флинна. Основы межпроцессорных взаимодействий. Основные узлы вычислительной системы. Микропроцессор. Память. Устройства ввода-вывода. Системная шина.

Раздел 2. БАЗОВАЯ АРХИТЕКТУРА ПРОЦЕССОРА

История развития процессоров Intel, AMD. Архитектура и характеристики микропроцессоров x86 серии. Архитектуры CISC и RISC. Архитектуры VLIW и EPIC. Модель микропроцессора для программиста. Обзор уровня архитектуры команд. Системные регистры микропроцессора. Форматы команд.

Раздел 3. РАСШИРЕНИЕ АРХИТЕКТУРЫ ПРОЦЕССОРА

Архитектура математического сопроцессора. Типы данных математического сопроцессора. Технология MMX. Технология SSE. Регистры MMX/XMM, типы данных и команды MMX/XMM.

Раздел 4. АРХИТЕКТУРА МНОГОЯДЕРНОГО ПРОЦЕССОРА

Структура современного многоядерного процессора. Состав и принцип работы ядра современного процессора. Способы повышения производительности ядра современного процессора. Архитектура процессоров ARM.

Раздел 5. МАТЕРИНСКАЯ ПЛАТА

Основные компоненты материнской платы. Чипсет: северный и южный мост. Основные шины. Подключение устройств к материнской плате.

Раздел 6. СИСТЕМНЫЕ УСТРОЙСТВА

Системные ресурсы ПЭВМ. Системный CMOS. Часы реального времени. Организация таймера ПЭВМ. Управление питанием и энергопотреблением. Подсистема ACPI.

Раздел 7. БАЗОВАЯ СИСТЕМА ВВОДА-ВЫВОДА

Функции BIOS. Процедура POST. Звуковые сигналы и текстовые сообщения. Главная загрузочная запись (MBR). Первичные и расширенные

разделы. UEFI. Предпосылки возникновения и основные отличия UEFI. Разбиение жесткого диска по схеме GPT.

Раздел 8. ПОДСИСТЕМА ПРЕРЫВАНИЙ

Прерывания в ПЭВМ: типы прерываний, их источники и приоритеты. Контроллер прерываний. Каскадное подключение контроллеров прерываний. Организация системы прерываний в современных ПК: расширенный контроллер прерываний APIC и технология MSI.

Раздел 9. ПОДСИСТЕМА ВВОДА-ВЫВОДА

Система ввода-вывода ПЭВМ. Управление вводом-выводом. Блочные и символьные операции. Синхронные и асинхронные операции. Отображение ввода-вывода на адресное пространство памяти. Кэширование операций. Упреждающее чтение. Отложенная запись. Программное обеспечение ввода-вывода.

Раздел 10. ПРЯМОЙ ДОСТУП К ПАМЯТИ

Технология прямого доступа к памяти (ПДП) и ее реализация в ПЭВМ. Контроллер ПДП: назначение, состав и режимы работы. Современная реализация технологии прямого доступа к памяти.

Раздел 11. ФАЙЛОВАЯ СИСТЕМА

Понятие файла. Назначение файловой системы. Принципы построения файловой системы. Файловые системы FAT и NTFS, их особенности и области использования.

Раздел 12. ПОДСИСТЕМА ПАМЯТИ

Иерархия памяти вычислительной системы. Кэш-память. Схемы организации оперативной памяти. Виртуальная память. Сегментная и страничная организации памяти. Таблицы страниц. Файл подкачки. Контроллер памяти.

Раздел 13. ДИСКОВАЯ ПОДСИСТЕМА

Принцип записи на магнитный диск. Организация информации на магнитных дисках: цилиндр, головка, сектор. LBA-адресация. Состав и функционирование накопителя на жестких магнитных дисках (НЖМД). Характеристики НЖМД. Контроллер НЖМД. Интерфейсы НЖМД.

Раздел 14. ВИДЕОСИСТЕМА

Назначение, состав и характеристики видеосистемы. Состав, функционирование и характеристики графического адаптера. Принципы работы и характеристики мониторов.

Раздел 15. ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА

Устройства ввода и устройства вывода информации. Клавиатура. Контроллер клавиатуры. Принтеры. Технологии печати. Последовательный и параллельный интерфейсы подключения периферийных устройств.

Лабораторные занятия

1. “Создание приложений взаимодействия с устройствами ввода-вывода”:
   - изучение программных средств разработки приложений в ОС Linux без графической оболочки;
   - изучение доступа к аппаратному обеспечению через файлы устройств в каталоге ./dev/ и с помощью системной функции ioperm();
   - разработка и реализация простейшего модуля ядра.
2. “Вывод информации об установленном оборудовании”:
   - изучение структуры системной области CMOS;
   - изучение способов чтения и записи данных в CMOS;
   - изучение макросов C для работы с портами ввода/вывода, комбинирования кода на языках C и ASM;
   - разработка и реализация алгоритма определения и вывода информации об установленном оборудовании.
3. “Переопределение кодов карты символов клавиатуры”
   - изучение различных вариантов переопределения кодов карты символов;
   - переопределение кодов с помощью командной строки;
   - переназначение кодов в графической оболочке.
4. “Работа с контроллером клавиатуры по прерыванию таймера”:
   - изучение команд контроллера клавиатуры;
   - разработка алгоритмов выполнения событий по таймеру с помощью CRON;
   - изучение работы с портами ввода/вывода, используя asm-вставки;
   - разработка и реализация алгоритма управления контроллером клавиатуры для эмуляции нажатой клавиши.
5. “Изучение обработчиков системных вызовов”
   - изучение назначение и состава таблицы прерываний;
   - изучение структуры системных файлов, хранящих версию ядра и адреса таблиц системных вызовов;
   - разработка алгоритмов определения адреса таблицы системных вызовов;
6. “Замена обработчика системного вызова в таблице прерываний во время компиляции”:

- изучение способов определения адреса таблицы системных вызовов во время компиляции;
- реализация алгоритма переопределения системного вызова open() на функцию вывода имени открываемого файла во время компиляции;

7. “Замена обработчика системного вызова во время работы модуля”:
   - изучение способов определения адреса таблицы системных вызовов во время работы модуля;
   - изучение функций чтения/записи в файл в режиме ядра;
   - реализация алгоритма переопределения системного вызова open() во время работы модуля;
8. “Работа со звуковыми устройствами”:
   - изучение способов воспроизведения звуков: с помощью таймера, с помощью файлов устройств ./dev/mixer/ и ./dev/dsp/;
   - изучение способов программирования звуковых устройств с помощью ./dev/dsp/;
   - изучение способов программирования звуковых устройств с использованием системного вызова ioctl();

Техническое обеспечение для организации лабораторных занятий

1. Учебный класс рабочих станций под управлением ОС Linux;
2. Файловый менеджер Midnight Commander, текстовый редактор mcedit, компилятор GCC, входящие в состав проекта GNU.

Самостоятельная работа студента

Самостоятельная работа студента над материалом дисциплины (без непосредственного управления преподавателем) выполняется в течение третьего семестра и включает в себя подготовку к лекционным и лабораторным занятиям, подготовку к сдаче экзамена, а также самостоятельное рассмотрение вопроса «Архитектура процессоров семейства ARM» из темы 7 «Способы повышения производительности ядра современного процессора» (в разделе 4 Архитектура многоядерного процессора).

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА ДИСЦИПЛИНЫ

Номер **раздела** / темы	Название раздела, темы	Количество аудиторных часов		Самостоятельная работа студента	Форма контроля знаний
		Лекции	Лабораторные занятия		
1	2	3	4	5	6
	Введение	**2**			
–, 1	История развития вычислительных систем. Понятие архитектуры персонального компьютера. Гарвардская и принстонская архитектура.	2			
1	**Архитектура вычислительных систем. Общая структура персональной ЭВМ**	**2**			
1 / 2	Общие сведения о ПЭВМ. Основные узлы. Микропроцессор. Кэш-память. ОЗУ. ПЗУ. Устройства ввода-вывода. Типы шин.	2			а)
2	**Базовая архитектура процессора**	**4**		**14**	
2 / 3	Архитектура и характеристики микропроцессоров x86 серии. Архитектура CISC, RISC, VLIW и EPIC. Модель микропроцессора для программиста. Обзор уровня архитектуры команд. Системные регистры микропроцессора. Форматы команд.	2			а), б)
2 / 4	История развития процессоров Intel, AMD.	2			
3	**Расширение архитектуры процессора**	**2**		**6**	
3 / 5	Архитектура математического сопроцессора. Типы данных математического сопроцессора. Технология MMX. Технология SSE. Регистры MMX/XMM, типы данных, команды.	2			а), б)
4	**Архитектура многоядерного процессора**	**4**		**6**	
4 / 6	Структура современного многоядерного процессора. Состав и принцип работы ядра современного процессора.	2			а)

4 / 7	Способы повышения производительности ядра современного процессора. Архитектура процессоров семейства ARM.	2		2	а)
5	**Материнская плата**	**2**		**8**	
5 / 8	Основные компоненты материнской платы. Чипсет: северный и южный мост. Типы шин. Подключение устройств к материнской плате.	2			а)
6	**Системные устройства**	**2**	**4**	**8**	
6 / 9	Системные ресурсы ПЭВМ. Системный CMOS. Часы реального времени. Организация таймера. Управление питанием и энергопотреблением. Подсистема ACPI.	2	4		а), б)
7	**Базовая система ввода-вывода**	**4**	**4**	**8**	
7 / 10	Функции BIOS. Процедура POST. Звуковые сигналы и текстовые сообщения. Главная загрузочная запись (MBR). Первичные и расширенные разделы.	2	4		а), б)
7 / 11	UEFI. Предпосылки возникновения и основные отличия. Разбиение жесткого диска по GPT.	2			а)
8	**Подсистема прерываний**	**4**	**12**	**6**	
8 / 12	Прерывания в ПЭВМ: типы прерываний, их источники и приоритеты. Таблица векторов прерываний. Контроллер прерываний. Каскадное подключение контроллера.	2	8		а), б)
8 / 13	Организация системы прерываний в современных ПК: расширенный контроллер прерываний APIC и технология MSI.	2	4		а), б)
9	**Подсистема ввода-вывода**	**2**	**8**	**6**	
9 / 14	Управление вводом-выводом. Блочные и символьные операции. Отображение ввода-вывода на адресное пространство памяти. Программное обеспечение ввода-вывода.	2	8		б)
10	**Прямой доступ к памяти**	**2**			
10 / 15	Реализация ПДП. Контроллер ПДП: состав, режимы работы. Современная реализация.	2			а)

1	2	3	4	5	6
11	**Файловая система**	**2**			
11 / 16	Понятие файла. Назначение файловой системы. Файловые системы FAT и NTFS.	2			а)
12	**Подсистема памяти**	**4**		**10**	
12 / 17	Иерархия памяти. Кэш-память. Типы кэш-памяти. Контроллер кэш-памяти.	2			а)
12 / 18	Схемы организации оперативной памяти. Виртуальная память, сегментная и страничная организации. Таблица страниц. Подкачка.	2			а)
13	**Дисковая подсистема**	**4**		**10**	
13 / 19	Состав и функционирование накопителя на жестких магнитных дисках (НЖМД). Характеристики НЖМД. Контроллер НЖМД.	2			а)
13 / 20	Принцип записи на магнитный диск. Организация информации: цилиндр, головка, сектор. LBA-адресация. Интерфейсы НЖМД.	2			а)
14	**Видеосистема**	**4**		**12**	
14 / 21	Назначение, состав и характеристики видеосистемы. Состав, функционирование и характеристики графического адаптера.	2			а)
14 / 22	Состав, характеристики и принцип работы ЭЛТ- и ЖК-мониторов.	2			а)
15	**Периферийные устройства**	**4**	**4**	**6**	
15 / 23	Устройства ввода. Принцип работы клавиатуры. Контроллеры. Интерфейсы подключения.	2	4		а), б)
15 / 24	Устройства вывода. Принтеры. Технологии печати. Интерфейсы подключения.	2			а)
Итого:		48	32	100	в)

Формы контроля знаний: а) контрольные опросы, б) защита лабораторной работы, в) экзамен.

4.2. ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

1. Орлов С.А., Цилькер Б.Я. Организация ЭВМ и систем: учебник для вузов. 2-е изд. – СПб.: Питер, 2011. — 688 с.

2. Танненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. – Спб.: Питер, 2019. - 816с.

3. Рудометов Е.А. Современное железо. Настольные, мобильные и встраиваемые компьютеры – СПб.: БХВ-Петербург, 2010. – 464 с.

4. Сидоров В.Д., Струмпэ Н.В. Аппаратное обеспечение ЭВМ. 3-е изд. – М.: 2014. –336 с.

5. Белунцов В. Секреты BIOS. – СПб.: Питер, 2005. — 336 с.

6. Зозуля Ю. Настройка компьютера с помощью BIOS. — СПб.: Питер, 2014. – 288 с.

7. Соломенчук В.Г., Соломенчук П.В. Железо ПК – СПб.: БХВ-Петербург, 2012. – 384 с.

8. Расширения традиционной архитектуры Intel // Системное программирование: учебно-методическое пособие В.Б. Синицина [Электронный ресурс]. – 2008. Режим доступа: https://studfile.net/preview/5352085/page:19/ Дата обращения: 30.08.2021

9. Расширение набора команд SSE4 для архитектуры Intel // По материалам документации Intel. Ализар А. [Электронный ресурс]. – 2008. Режим доступа: https://www.kv.by/archive/index2008010503.htm. Дата обращения: 30.08.2021

10. Intel — история успеха // Хабр. Лис А. [Электронный ресурс]. – 2017. Режим доступа: https://habr.com/ru/post/406029/ Дата обращения: 30.08.2021

11. Эволюция процессоров Intel: от Core 2 Duo до Core i9 [Электронный ресурс]. – 2017. Режим доступа: https://ek.ua/post/1394/186-evolyutsiya-protsessorov-intel-ot-core-2-duo-do-core-i9/ Дата обращения: 30.08.2021

12. Гук М.Ю. Аппаратные средства IBM PC – СПб.: Питер, 2006. – 1072 с.

13. Дроздов С. Операционные системы. Учебное пособие. – Ростов-на-Дону: Феникс, 2016 – 361 с.

14. Сергеев С.Л. Архитектуры вычислительных систем – СПб.: БВХ-Петербург, 2010. – 240 с.

15. Фролов А. Аппаратное обеспечение персонального компьютера. – М.: Диалог-МИФИ, 1997, 304 с.

16. Фролов А. Программирование видеоадаптеров. – М.: Диалог-МИФИ, 1992, 287 с.