ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ Б1.Б.1.27 Организация ЭВМ и вычислительных систем

Направление подготовки: 10.05.03 Информационная безопасность автоматизированных систем

Специализация: «Информационная безопасность автоматизированных систем критически важных объектов»

Форма обучения: очная

Содержание

1. КОНСПЕКТ ЛЕКЦИЙ

1. 1 Лекция №1-2 (4 часа).

Тема: «Арифметические основы построения и логические основы построения ЭВМ»

1.1.1 Вопросы лекции:

1) История возникновения и развития вычислительной техники
2) Классификация компьютеров. Поколения вычислительной техники

1.1.2 Краткое содержание вопросов:

1) История возникновения и развития вычислительной техники

Первым устройством, предназначенным для облегчения счета, были счеты. С помощью костяшек счетов можно было совершать операции сложения и вычитания и несложные умножения.

1642 г. — французский математик Блез Паскаль сконструировал первую механическую счетную машину «Паскалина», которая могла механически выполнять сложение чисел.

1673 г. — Готфрид Вильгельм Лейбниц сконструировал арифмометр, позволяющий механически выполнять четыре арифметических действия.

Первая половина XIX в. — английский математик Чарльз Бэббидж попытался построить универсальное вычислительное устройство, то есть компьютер. Бэббидж называл его аналитической машиной. Он определил, что компьютер должен содержать память и управляться с помощью программы. Компьютер по Бэббиджу — это механическое устройство, программы для которого задаются посредством перфокарт — карт из плотной бумаги с информацией, наносимой с помощью отверстий (они в то время уже широко употреблялись в ткацких станках).

1941 г. — немецкий инженер Конрад Цузе построил небольшой компьютер на основе нескольких электромеханических реле.

1943 г. — в США на одном из предприятий фирмы IBM ГовардЭйкен создал компьютер под названием «Марк-1». Он позволял проводить вычисления в сотни раз быстрее, чем вручную (с помощью арифмометра), и использовался для военных расчетов. В нем использовалось сочетание электрических сигналов и механических приводов. «Марк-1» имел размеры: 15 * 2—5 м и содержал 750 000 деталей. Машина была способна перемножить два 32-разрядных числа за 4 с.

1943 г. - в США группа специалистов под руководством Джона Мочли и ПроспераЭкерта начала конструировать компьютер ENIAC на основе электронных ламп.

1945 г. — к работе над ENIAC был привлечен математик Джон фон Нейман, который подготовил доклад об этом компьютере. В своем докладе фон Нейман сформулировал

общие принципы функционирования компьютеров, т. е. универсальных вычислительных устройств. До сих пор подавляющее большинство компьютеров сделано в соответствии с теми принципами, которые изложил Джон фон Нейман.

1947 г. — Экертом и Мочли начата разработка первой электронной серийной машины UNIVAC (UniversalAutomaticComputer). Первый образец машины (UNIVAC-1) был построен для бюро переписи США и пущен в эксплуатацию весной 1951 г. Синхронная, последовательного действия вычислительная машина UNIVAC-1 была создана на базе ЭВМ ENIAC и EDVAC. Работала она с тактовой частотой 2,25 МГц и содержала около 5000 электронных ламп. Внутреннее запоминающее устройство емкостью 1000 12-разрядных десятичных чисел было выполнено на 100 ртутных линиях задержки.

1949 г. — английским исследователем Морнсом Уилксом построен первый компьютер, в котором были воплощены принципы фон Неймана.

1951 г. — Дж. Форрестер опубликовал статью о применении магнитных сердечников для хранения цифровой информации, В машине «Whirlwind-1» впервые была применена память на магнитных сердечниках. Она представляла собой 2 куба с 32-32-17 сердечниками, которые обеспечивали хранение 2048 слов для 16-разрядных двоичных чисел с одним разрядом контроля на четность.

1952 г. — фирма IBM выпустила свой первый промышленный электронный компьютер IBM 701, который представлял собой синхронную ЭВМ параллельного действия, содержащую 4000 электронных ламп и 12 000 диодов. Усовершенствованный вариант машины IBM 704 отличался высокой скоростью работы, в нем использовались индексные регистры и данные представлялись в форме с плавающей запятой.

После ЭВМ IBM 704 была выпущена машина IBM 709, которая в архитектурном плане приближалась к машинам второго и третьего поколений. В этой машине впервые была применена косвенная адресация и впервые появились каналы ввода — вывода.

1952 г. — фирма RemingtonRand выпустила ЭВМ UNIVAC-t 103, в которой впервые были применены программные прерывания. Сотрудники фирмы RemingtonRand использовали алгебраическую форму записи алгоритмов под названием «ShortCode» (первый интерпретатор, созданный в 1949 г. Джоном Мочли).

1956 г. — фирмой IBM были разработаны плавающие магнитные головки на воздушной подушке. Изобретение их позволило создать новый тип памяти — дисковые запоминающие устройства (ЗУ), значимость которых была в полной мере оценена в последующие десятилетия развития вычислительной техники. Первые ЗУ на дисках появились в машинах IBM 305 и RAMAC. Последняя имела пакет, состоявший из 50 металлических дисков с магнитным покрытием, которые вращались со скоростью 12000 об. /мин. На поверхности диска размещалось 100 дорожек для записи данных, по 10 000 знаков каждая.

1956 г. — фирма Ferranti выпустила ЭВМ «Pegasus», в которой впервые нашла воплощение концепция регистров общего назначения (РОН). С появлением РОН

устранено различие между индексными регистрами и аккумуляторами, и в распоряжении программиста оказался не один, а несколько регистров-аккумуляторов.

1957 г. — группа под руководством Д. Бэкуса завершила работу над первым языком программирования высокого уровня, получившим название ФОРТРАН. Язык, реализованный впервые на ЭВМ IBM 704, способствовал расширению сферы применения компьютеров.

1960-е гг. — 2-е поколение ЭВМ, логические элементы ЭВМ реализовываются на базе полупроводниковых приборов-транзисторов, развиваются алгоритмические языки программирования, такие как Алгол, Паскаль и другие.

1970-е гг. — 3-е поколение ЭВМ, интегральные микросхемы, содержащие на одной полупроводниковой пластине тысячи транзисторов. Начали создаваться ОС, языки структурного программирования.

1974 г. — несколько фирм объявили о создании на основе микропроцессора Intel-8008 персонального компьютера — устройства, выполняющего те же функции, что и большой компьютер, но рассчитанного на одного пользователя.

1975 г. — появился первый коммерчески распространяемый персональный компьютер Альтаир-8800 на основе микропроцессора Intel-8080. Этот компьютер имел оперативную память всего 256 байт, клавиатура и экран отсутствовали.

Конец 1975 г. — Пол Аллен и Билл Гейтс (будущие основатели фирмы Microsoft) создали для компьютера «Альтаир» интерпретатор языка Basic, позволивший пользователям просто общаться с компьютером и легко писать для него программы.

Август 1981 г. — компания IBM представила персональный компьютер IBM PC. В качестве основного микропроцессора компьютера использовался 16-разрядный микропроцессор Intel-8088, который позволял работать с 1 мегабайтом памяти.

1980-е гг. — 4-е поколение ЭВМ, построенное на больших интегральных схемах. Микропроцессоры реализовываются в виде единой микросхемы, Массовое производство персональных компьютеров.

1990-е гг. — 5-е поколение ЭВМ, сверхбольшие интегральные схемы. Процессоры содержат миллионы транзисторов. Появление глобальных компьютерных сетей массового пользования.

2000-е гг. — 6-е поколение ЭВМ. Интеграция ЭВМ и бытовой техники, встраиваемые компьютеры, развитие сетевых вычислений.

2) Классификация компьютеров. Поколения вычислительной техники

Компьютеры могут быть классифицированы по разным признакам, например по габаритам, по областям применения, по быстродействию, по функциям, по этапам создания и еще по многим другим параметрам.

Мы рассмотрим классификацию по обобщенному параметру, где в разной степени учтено несколько характерных признаков:

назначение и роль компьютеров в системе обработки информации;

условия взаимодействия человека и компьютера;

габариты компьютера;

ресурсные возможности компьютера.

В соответствии с вышесформулированными признаками и тенденциями развития компьютерной техники можно рассмотреть следующую классификацию компьютеров:
портативные компьютеры;
микрокомпьютеры, в том числе — персональные компьютеры;
мэйнфреймы (универсальные компьютеры);
суперкомпьютеры.
Портативные компьютеры обычно нужны руководителям предприятий, менеджерам, учёным, журналистам, которым приходится работать вне офиса — дома, на презентациях или во время командировок.
Основные разновидности портативных компьютеров:

Микрокомпьютеры — это компьютеры, в которых центральный процессор выполнен в виде микропроцессора.
Продвинутые модели микрокомпьютеров имеют несколько микропроцессоров.
Производительность компьютера определяется не только характеристиками применяемого микропроцессора, но и ёмкостью оперативной памяти, типами периферийных устройств, качеством конструктивных решений и др.
Микрокомпьютеры представляют собой инструменты для решения разнообразных сложных задач. Их микропроцессоры с каждым годом увеличивают мощность, а периферийные устройства — эффективность. Быстродействие — порядка 1 - 10 миллионов операций в сек.

Разновидность микрокомпьютера — микроконтроллер. Это основанное на микропроцессоре специализированное устройство, встраиваемое в систему управления или технологическую линию.
В класс персональных компьютеров входят различные машины — от дешёвых домашних и игровых с небольшой оперативной памятью, с памятью программы на кассетной ленте и обычным телевизором в качестве дисплея, до сверхсложных машин с мощным процессором, винчестерским накопителем ёмкостью в десятки Гигабайт, с цветными графическими устройствами высокого разрешения, средствами мультимедиа и другими дополнительными устройствами.

Персональныйкомпьютер должен удовлетворять следующим требованиям:
стоимость от нескольких сотен до 5-10 тысяч долларов;
наличие внешних ЗУ на магнитных дисках;
объём оперативной памяти не менее 4 Мбайт;
наличие операционной системы;
способность работать с программами на языках высокого уровня;
ориентация на пользователя-непрофессионала (в простых моделях).

Мейнфрейм - это синоним понятия "большая универсальная ЭВМ". Мейнфреймы и до сегодняшнего дня остаются наиболее мощными (не считая суперкомпьютеров)вычислительными системами общего назначения, обеспечивающими непрерывный круглосуточный режим эксплуатации. Они могут включать один или несколько процессоров, каждый из которых, в свою очередь, может оснащаться векторными сопроцессорами (ускорителями операций с суперкомпьютерной производительностью). В нашем сознании мейнфреймы все еще ассоциируются с большими по габаритам машинами, требующими специально оборудованных помещений с системами водяного охлаждения и кондиционирования. Однако это не совсем так. Прогресс в области элементно-конструкторской базы позволил существенно сократить габариты основных устройств. Наряду со сверхмощными мейнфреймами, требующими организации двухконтурной водяной системы охлаждения, имеются менее мощные модели, для охлаждения которых достаточно принудительной воздушной вентиляции, и модели, построенные по блочно-модульному принципу и не требующие специальных помещений и кондиционеров.

Основными поставщиками мейнфреймов являются известные компьютерные компании IBM, Amdahl, ICL, SiemensNixdorf и некоторые другие, но ведущая роль принадлежит безусловно компании IBM. Именно архитектура системы IBM/360, выпущенной в 1964 году, и ее последующие поколения стали образцом для подражания. В нашей стране в течение многих лет выпускались машины ряда ЕС ЭВМ, являвшиеся отечественным аналогом этой системы. В архитектурном плане мейнфреймы представляют собоймногопроцессорные системы, содержащие один или несколько центральных и периферийных процессоров с общей памятью, связанных между собой высокоскоростными магистралями передачи данных. При этом основная вычислительная нагрузка ложится на центральные процессоры, а периферийные процессоры (в терминологии IBM - селекторные, блок-мультиплексные, мультиплексные каналы и процессоры телеобработки) обеспечивают работу с широкой номенклатурой периферийных устройств.
Они предназначены для решения широкого класса научно-технических задач и являются сложными и дорогими машинами. Их целесообразно применять в больших системах при наличии не менее 200 - 300 рабочих мест.
Централизованная обработка данных на мэйнфрейме обходится примерно в 5 - 6 раз дешевле, чем распределённая обработка при клиент-серверном подходе.
Известный мейнфрейм S/390 фирмы IBM обычно оснащается не менее чем тремя процессорами. Максимальный объём оперативного хранения достигает 342 Терабайт. Производительность его процессоров, пропускная способность каналов, объём оперативного хранения позволяют наращивать число рабочих мест в диапазоне от 20 до 200000 с помощью простого добавления процессорных плат, модулей оперативной памяти и дисковых накопителей.
Десятки мейнфреймов могут работать совместно под управлением одной операционной системы над выполнением единой задачи.
Отличительной особенностью суперкомпьютеров являются векторные процессоры, оснащенные аппаратурой для параллельного выполнения операций с многомерными цифровыми объектами — векторами и матрицами. В них встроены векторные регистры и параллельный конвейерный механизм обработки. Если на обычном процессоре программист выполняет операции над каждым компонентом вектора по очереди, то на векторном — выдаёт сразу векторые команды.
Векторная аппаратура очень дорога, в частности, потому, что требуется много сверхбыстродействующей памяти под векторные регистры.
Наряду с векторно-конвейерной системой обработки данных существует и скалярная система, основанная на выполнении обычных арифметических операций над отдельными числами или парами чисел.
Строго говоря, системы, использующие скалярную обработку данных, по своей производительности уступают суперЭВМ, но у них наблюдаются тенденции, характерные для высокопроизводительных вычислительных систем: необходимость распараллеливания больших задач между процессорами.
Наиболее распространённые суперкомпьютеры — массово-параллельные компьютерные системы. Они имеют десятки тысяч процессоров, взаимодействующих через сложную, иерархически организованую систему памяти.

Супер-компьютеры используются для решения сложных и больших научных задач (метеорология, гидродинамика и т. п.), в управлении, разведке, в качестве централизованных хранилищ информации и т.д.
Элементная база — микросхемы сверхвысокой степени интеграции.
К суперкомпьютерам часто относят и серверы.

Сервер-мощный компьютер в вычислительных сетях, который обеспечивает обслуживание подключенных к нему компьютеров и выход в другие сети.
В зависимости от назначения определяют такие типы серверов:

Сервер приложений обрабатывает запросы от всех станций вычислительной сети и предоставляет им доступ к общим системным ресурсам (базам данных, библиотекам программ, принткрам, факсам и др.).
Файл-сервер-для работы с базами данных и использования файлов информации, хранящихся в ней.
Архивационный сервер-для резервного копирования информации в крупных многосервисных сетях. Он использует накопители на магнитной ленте(стриммеры) со сменными картриджами емкостью до 5 Гбайт. Обычно выполняет ежедневное автоматическое архивирование информации от подключенных серверов и рабочих станций.
Факс-сервер-для организации эффективной многоадресной факсимильной связи, с несколькими факсмодемными платами, со специальной защитой информации от несанкционированного доступа в процессе передачи, с системой хранения электронных факсов.
Почтовый сервер-то же, что и факс-сервер, но для организации электронной почты, с электронными почтовыми ящиками.
Сервер печати-для эффективного использования системных принтеров.
Сервер- телеконференций-компьютер, имеющий программу обслуживания пользователей телеконференциями и новостями, он также может иметь систему автоматической обработки видеоизображений и др.
Любой компьютер, если установить на нем соответствующее сетевое программное обеспечение, способен стать сервером. Кроме того, один компьютер одновременно может выполнять несколько функций-быть, к примеру, почтовым сервером, сервером новостей, сервером приложений и т.д.
Существуют различные классификации компьютерной техники:
по этапам развития (по поколениям);
по архитектуре;
по производительности;
по условиям эксплуатации;
по количеству процессоров;
по потребительским свойствам и т.д.
Четких границ между классами компьютеров не существует. По мере совершенствования структур и технологии производства, появляются новые классы компьютеров, границы существующих классов существенно изменяются.

1. 2 Лекция №3-4(4 часа).
Тема: «Минимизация логических функций. Выполнение операций в двоичном коде**»**

1.2.1 Вопросы лекции:

1) Архитектура и структура компьютера
2) Состав и назначение основных устройств ЭВМ

1.2.2 Краткое содержание вопросов:

1) Архитектура и структура компьютера

2)Архитектура – это совокупность характеристик и свойств структуры и программных средств с точки зрения пользователя. Т.е. архитектура ПК связана с особенностью его построения и функционирования как аппаратно-программного комплекса.
С целью унификации и упрощения задач управления и обслуживания ПК его аппаратные и программные средства компонуются в виде отдельных модулей.
Компьютер состоит из центральной части (одного или нескольких процессоров,

выполняющих преобразование информации, и оперативной памяти), средств связи (каналов) и периферийных устройств (устройств ввода-вывода, отображения и запоминания информации).
Каждая из этих частей может содержать:
аппаратные средства - электронные узлы и блоки, реализованные с использованием электронных микросхемы малой, средней, большой и сверхбольшой интеграции (МИС, СИС, БИС, СБИС), электрические и механические узлы (устройства, разъемы);
микропрограммные средства - записанные в постоянную память программы управления передачей команд и данных в различных узлах ПК;
программные средства - системные программы, операционные системы среды, различные утилиты, прикладные программы.

Так, в настоящее время семейство ПК IBM включает более 200 модулей объединяемых в единое целое системой шин (магистралей).
Основная задача шин - объединить в единую систему разнообразную номенклатуру модулей, обеспечив их высокопроизводительную надлежащую работу. Под надлежащей работой следует понимать выполнение условий:
открытости (возможность расширять и модернизировать);
совместимости (взаимозаменяемость);
однотипности;
гибкости (возможность подключения различных подсистем);
надежности;
ремонтопригодности;
эффективности (экономическая целесообразность) и др. общесистемных требований.

Эффективная работа ПК связана с надлежащим согласованием между микропроцессором, памятью и коммуникационными магистралями.
Современная типовая структура системы шип в ПК IBM PC AT включает:
локальную шину (L - Localbus), к которой подключается микропроцессор;
локальную шину памяти (M - Memorybus), к которой подключается оперативная память;
системную шину (S - Systembus), связывающую работу всех модулей компьютера в единое целое;
- внешнюю (периферийную) шину (X - externalbus), связанную с периферийными модулями.

2) Состав и назначение основных устройств ЭВМ

Персональный компьютер (ПК) – комплекс взаимосвязанных устройств, каждому из которых поручена определенная функция – это системный блок, монитор (дисплей), клавиатура, мышь, соединенные кабелями или беспроводной связью.

В системном блоке расположены основные аппаратные компоненты ПК:

— материнская (системная) плата;
— процессор;
— память;
— адаптеры (контроллеры) внешних устройств;
— дисководы для гибких и оптических дисков;
— дисководы на жестком магнитном диске («винчестеры»);
— органы управления (выключатели, кнопка сброса, индикаторы питания и режимов работы).

Каждый из функциональных элементов (память, монитор или другое устройство) связан с шиной определенного типа – *адресной, управляющей* или *шиной данных*. Для согласования

интерфейсов периферийные устройства подключаются к шине через контроллеры и порты.

Контроллер – специализированное устройство (или плата), управляющее работой некоторого периферийного устройства и обеспечивающее его связь с системной платой. Например, контроллер клавиатуры или жёсткого диска.

Адаптер – устройство, обеспечивающее согласование параметров входных и выходных сигналов в системе. Например, видеоадаптер, преобразующий цифровое изображение для отображения на аналоговом мониторе; адаптеры последовательного и параллельного портов.

Порты устройств – электронные схемы, содержащие один или несколько регистров ввода-вывода и позволяющие подключать периферийные устройства компьютера к внешним шинам процессора.

Последовательный порт обменивается данными с процессором побайтно, а с внешними устройства побитно.

Параллельный порт получает и посылает данные побайтно.

К последовательному порту подключают медленно действующие или достаточно удаленные устройства (мышь, модем). К параллельному порту подсоединяют более быстрые устройства (принтер, сканер).

На материнской плате расположены главные компоненты компьютерной системы:

- центральный микропроцессор, совмещающий в современных ПК АЛУ и УУ;
- оперативная память;
- микросхемы поддержки;
- центральная магистраль или шина;
- контроллер шины и несколько гнезд-разъемов (слотов).

Они служат для подключения к материнской плате других устройств.

Существуют следующие *системные* шины:

- 16-разрядная шина ISA, работающая с тактовой частотой 8 МГц;
- шины MCA с тактовой частотой до 10 МГц, разрядность шины 16 и 32 (шина несовместима с шиной ISA , поэтому практически не используется);
- шина EISA работает с тактовой частотой 8-10 МГц (применялась в высокопроизводительных серверах и профессиональных рабочих станциях), являлась очень дорогим решением.

Для увеличения производительности используют **локальные шины**, связывающие процессор непосредственно с контроллерами периферийных устройств. Наиболее известные – VL-bus, PCI и PCI-Express. Для ускорения работы с графическими и видеоданными шина PCI дополнена портом AGP.

Высокоскоростные последовательные шины USB – служит для одновременного подключения большого количества внешних устройств – до 127, и IEEE-1394 (FireWire) –

используется для соединения с внешними проигрывателями дисков DVD и CD-ROM, с цифровыми видео- и фотокамерами, с жесткими дисками.

Основные характеристики процессора – **тактовая частота** – количество элементарных операций, выполняемых процессором, в секунду и**разрядность** – количество двоичных разрядов, обрабатываемых за один такт, и количество разрядов, используемых для адресации оперативной памяти.

1. 3 Лекция №5-6 (4 часа).

Тема: «Построение логических схем. Комбинированные узлы»

1.3.1 Вопросы лекции:

1) Организация памяти. Микросхемы памяти
2) Процессор. Функции, параметры, структура процессора

1.3.2 Краткое содержание вопросов:

1) Организация памяти. Микросхемы памяти

Производительность ПК зависит от типа и размера ОП, а это в свою очередь зависит от набора интегральных схем на материнской плате.
Внешний вид микросхем ОП: пластиковая полоска, на ней расположены кремневые «черепашки» – чипы-микросхемы (то есть используется полупроводниковая технология) и имеются «ножевые» контактные разъемы.
Устройства памяти характеризуются следующими основными показателями:
временем доступа (быстродействием). Время доступа – промежуток времени, за который может быть записано (прочитано) содержимое ячейки памяти.
емкостью (определяет количество ячеек (битов) в устройстве памяти).
стоимостью.
потребляемой мощностью (электропотреблением).
Существует 2 модуля памяти, отличающиеся формой, внутренней архитектурой, скоростью работы:SIMM и DIMM.
I. SIMM (SINGLE IN-LINE MEMORY MODULES) (SRAM)
бывают двух типов (отличающихся количеством контактов).
1. 30-контактные модули SIMM. Бывают 1 и 4 Мб. Практически сегодня исчезли из продажи для компьютеров 386, 286-процессором. Сегодня им нашлось интересное применение – в качестве ОП, устанавливаемой в некоторые звуковые платы, например, GreafiveSoundBlaster 32 (AWE-32) GravisUltraSoundPnP. Однако новая карта AWE-64 уже содержит свои модули ОП, эта память не нужна.
2. 72-контактные SIMM (на 1, 4, 8, 16, 32, 64 Мб, редко 128 Мб). Внешний вид неизменный, а вот тип устанавливаемой на них памяти меняется (тип памяти указывается на микросхеме).
a) самый старый (редко сейчас встречающийся) – FPM DRAM (или просто DRAM – DynamicRandomAccessMemory – динамическая ОП). Работала на 486 и первых Pentium.
b) модифицированный тип EDO DRAM (или EDO – Extendeddataoutput).
Микросхемы SIMM выпускаются одинарной и двойной плотности, с контролем четности и без (использование контроля четности позволяет парировать одиночную ошибку памяти). Модули отличаются и по скорости доступа 60 и 70 наносекунд, чем скорость меньше, тем быстрее доступ. 60 наносекунд быстрее 70 наносекунд. Модули SIMM в материнской плате Pentium и Pentium MMX устанавливаются только попарно, образуя так называемый банк.
Пример необходимо 32 Мб => 2 модуля SIMM по 16 Мб.
необходимо 64 Мб => 4 модуля SIMM по 16 Мб или 2 модуля SIMM по 32Мб.

В рамках одного банка можно использовать только одинаковые по емкости и скорости доступа модули SIMM. Если на вашей материнской плате 4 слота для модулей памяти SIMM, то можно сформировать два банка различной емкости.

II. DIMM (SDRAM DUAL IN-LINE MEMORY MODULES).
Появился впервые у MMX- компьютеров, стал основой для PII., поэтому у PII редко бывают SIMM-разъемы. DIMM не обязательно должно быть четное число. Модули DIMM бывают емкостью 16, 32, 64, 128, 256, 512 Мб
Виды DIMM.
EDO SD RAM (Synchronous DRAM) – синхронизируемая динамическая ОП)
SD RAM (SINGLE DATA RATE RANDOM ACCESS MEMORY).ЗУПВ с одинарной скоростью передачи данных, которая в зависимости от тактовой частоты называется памятью PC100 и PC133. Микросхема на 168 контактов, является сегодня самой «медленной» из семейства DIMM-модулей памяти, Время доступа = 10-20 наносекунд. Верхний предел ее тактовой частоты 133 МГц. И все же этот тип ОП вполне подходит для большинства офисных и
домашних ПК. Пропускная способность 1Гб/с.
SPD – это небольшая микросхема, установленная в модуле памяти SD RAM DIMM и содержащая подробную информацию о типе установленной памяти и некоторые другие устройства. PC133 SDRAM(SynchronousDynamicRandomAccessMemory) самая быстрая из класса классической ОП. (были и PC66, PC100). Теперь это самый медленный тип ОЗУ. Физически представляет собой массив микроскопических конденсаторов, «упакованных» в микросхемы памяти. Логически каждый конденсатор есть не что иное, как элементарная однобитовая информационная ячейка с 2-мя состояниями: 0 – если конденсатор не заряжен, 1 – если заряжен. Эти ячейки объединяются в двумерную матрицу, где каждая ячейка адресуется номерами строки и столбца, на пересечении которых она находится. К микросхеме подводятся шины командная (передает команды, управляющие работой микросхем ОП), адресная (адреса строк и столбцов), и данных. Все три синхронизируются импульсами одной и той же частоты. (133). SDRAM – синхронная память и логика работы микросхем памяти этого типа жестко синхронизируется с тактовым сигналом. Например, контроллер памяти точно знает, в течение скольких тактов микросхемы памяти будут готовить запрошенные данные для передачи и на каком такте начнется собственно их передача. Сегодня данная микросхема встречается редко.
Rambus (RD RAM)Двухканальная ОП (микросхема фирмы Intel). DirectRambus – это новая шина памяти, в которой управление адресацией отделено от работы с данными. Система состоит из контроллера DirectRambus, подсоединенного к одному или нескольким модулям DirectRambus DRAM, которые называются RIMM, в отличии от обычных микросхем памяти, соединяемых параллельно, RIMM соединяются последовательно. Канал DirectRambus включает двунаправленную шину данных и шину адреса, т.е. адреса памяти передаются одновременно с данными. Каждая микросхема RDRAM может содержать до 32 независимых банков, SD RAM – от 2 до 8. Свободно работает на высоких тактовых частотах.
Микросхема на 184 контакта Микросхемы ОП с тактовой частотой от 600 до 800 МГц. Когда используется микросхема PC800 (частота синхронизации 400 МГц), пропускная способность шины «память-процессор» достигает 3,2 Гб/с. При использовании PC600 (300 МГц) этот параметр = 2,6 Гб/с.
В свободные гнезда памяти Rambus необходимо устанавливать заглушки ContinuityRimm (CRIMM). Без них система не станет работать, поскольку модули в обоих каналах Rambus включаются каскадно, то есть тактовые и управляющие сигналы проходят через разъемы Rimm последовательно. Емкость ОЗУ может быть до 3 Гб.
Обеспечивают значительное быстродействие при выполнении сложных приложений на ПК и рабочих станциях. Вопрос о быстродействии ОП сегодня очень спорный.

DDR SDRAM (DoubleDataRate) – двойная скорость передачи данных – это по сути модификации обычной SDRAM и отличается от нее тем, что в ней запись и чтение данных происходят и по переднему и по заднему фронту тактового импульса. Поэтому за один такт по шине передается вдвое больше данных, и ее эффективная частота оказывается вдвое больше физической.
2х канальная память DDR266 DDR333 и DDR400 и системы с ней не уступают памяти RDRAM. ОП с удвоенной скоростью передачи данных, а иначе называется PC200 и PC266 в зависимости от тактовой частоты системной шины. Не столь дорогая, чем (3) и явно способствует повышению быстродействия ПК в отличие от (2). В основном благодаря использованию этой памяти ПК на базе Athlon 1,2 Ггц обошел на многих тестах 1,5 Ггц P-IV с памятью RD RAM.
Сегодня, пока, покупатель не может просто выбрать желательный для него тип ОП, так как она связана с интегральной схемой на системной плате, а та с ЦП. Так, пока, P-IV работает с набором ИС- 850 компании Intel и дорогостоящей памятью RD RAM. (В середине 2001 года планируется появление микросхем, совместимых с устройствами SD RAM и DDR). Если вы хотите приобрести P-IV, то автоматически будете вынуждены приобрести и дорогую ОП. Наборы интегральных схем семейства Athlon используют ОП SD RAM и DDR, но не могут RD RAM.

2) Процессор. Функции, параметры, структура процессора

Основные функции любого процессора следующие:
1)выборка (чтение) выполняемых команд;
2)ввод (чтение) данных из памяти или УВВ;
3)вывод (запись) данных в память или УВВ;
4)обработка данных (операндов), в том числе арифметические операции над ними;
5)адресация памяти, т. е. задание адреса памяти, с которым будет производиться обмен;
6)обработка прерываний и режима прямого доступа к памяти (ПДП).
Важнейшая характеристика процессора-разрядность.Разрядность ШД-скорость работы системы.Разрядность ША-допустимая сложность системы.Кол-во линий управления определяет разнообразие режимов обмена и эффективность обмена процессора с другими устройствами системы.
Магистраль (системная шина) включает в себя три многоразрядные шины:шину данных, шину адреса,шину управления.
Шина данных служит для пересылки данных между ЦП и памятью или ЦП и устройствами ввода/вывода.Шинаадреса .Выбор устройства или ячейки памяти, куда пересылаются или откуда считываются данные по шине данных, производит процессор.Каждое устройство или ячейка оперативной памяти имеет свой адрес. Адрес передается по адресной шине, причем сигналы по ней передаются в одном направлении - от процессора к оперативной памяти и устройствам (однонаправленная шина).
По шине управления передаются управляющие сигналы, определяющие характер обмена

информацией по магистрали и предназначенные памяти и устройствам ввода/вывода.
Микросхема процессора обязательно имеет выводы трех шин: шины адреса, шины данных и шины управления.CLK-подключение внешнего тактового сигнала и тактового резонатора (быстродействие CPU). RESET-сигнал начального сброса. Для подключения CPU к магистрали используют буферные микросхемы, обеспечивающие, если необходимо, демультиплексирвоание сигналов и электрическое буферирование сигналов магистрали. Буферные микросхемы согласуют протоколы шин процессора и магистрали если они не совпадают.
Внутренняя структура микропроцессора. Схема управления выборкой команд, АЛУ, регистры процессора, схема управления прерываниями, схема управления прямым доступом к памяти, логика управления.

Схема управления выборкой команд выполняет чтение команд из памяти и их дешифрацию.
Арифметико-логическое устройство (или АЛУ, ALU) предназначено для обработки информации в соответствии с полученной процессором командой.
Быстродействие АЛУ во многом определяет производительность процессора. Причем важна не только частота тактового сигнала, которым тактируется АЛУ, но и количество тактов, необходимое для выполнения той или иной команды.
Для повышения производительности разработчики стремятся довести время выполнения команды до одного такта, а также обеспечить работу АЛУ на возможно более высокой частоте. Другой путь повышения производительности процессора — использование нескольких параллельно работающих АЛУ.
Регистры процессора представляют собой по сути ячейки очень быстрой памяти и служат для временного хранения различных кодов: данных, адресов, служебных кодов. Операции с этими кодами выполняются предельно быстро, поэтому, в общем случае, чем больше внутренних регистров, тем лучше. Кроме того, на быстродействие процессора сильно влияет разрядность регистров. Именно
разрядность регистров и АЛУ называется внутренней разрядностью процессора, которая может не совпадать с внешней разрядностью.
По отношению к назначению внутренних регистров существует два основных подхода. Первого придерживается, например, компания Intel, которая каждому регистру отводит строго определенную функцию. С одной стороны, это упрощает организацию процессора и уменьшает время выполнения команды, но с другой — снижает гибкость, а иногда и замедляет работу программы. Второй подход состоит в том, чтобы все (или почти все)регистры сделать равноправными, как , например, в 16-разрядных процессорах Т-11 фирмы DEC. При этом достигается высокая гибкость, но необходимо усложнение структуры процессора.PSW-содержит информацию о выполнении пред-й команды.
Среди общих регистров имеются регистры специального назначения: указатель стека SP (StackPointer), счетчик команд PC (ProgramCounter)

Схема управления прерываниями обрабатывает поступающий на процессор запрос прерывания, определяет адрес начала программы обработки прерывания (адрес вектора прерывания), обеспечивает переход к этой программе после выполнения текущей команды и сохранения в памяти (в стеке) текущего состояния регистров процессора. По окончании программы обработки прерывания процессор возвращается к прерванной программе с восстановленными из памяти (изстека) значениями внутренних регистров.
Схема управления прямым доступом к памяти служит для временного отключения процессора от внешних шин и приостановки работы процессора на время предоставления прямого доступа запросившему его устройству.
Логика управления организует взаимодействие всех узлов процессора, перенаправляет данные, синхронизирует работу процессора с внешними сигналами, а также реализует процедуры ввода и вывода информации.

1. 4 Лекция №7-8 (4 часа).
Тема: «Узлы с памятью**»**

1.4.1 Вопросы лекции:

1) Введение в язык Ассемблера.
2) Язык Ассемблера. Основные команды.

1.4.2 Краткое содержание вопросов:

1) Введение в язык Ассемблера.

Для начала разберёмся с терминологией.
Машинный код – система команд конкретной вычислительной машины (процессора), которая интерпретируется непосредственно процессором. Команда, как правило,

представляет собой целое число, которое записывается в регистр процессора. Процессор читает это число и выполняет операцию, которая соответствует этой команде
Язык программирования низкого уровня (низкоуровневый язык программирования) – это язык программирования, максимально приближённый к программированию в машинных кодах. В отличие от машинных кодов, в языке низкого уровня каждой команде соответствует не число, а сокращённое название команды (мнемоника). Например, команда ADD – это сокращение от слова ADDITION (сложение). Поэтому использование языка низкого уровня существенно упрощает написание и чтение программ (по сравнению с программированием в машинных кодах). Язык низкого уровня привязан к конкретному процессору. Например, если вы написали программу на языке низкого уровня для процессора PIC, то можете быть уверены, что она не будет работать с процессором AVR.
Язык программирования высокого уровня – это язык программирования, максимально приближённый к человеческому языку (обычно к английскому, но есть языки программирования на национальных языках, например, язык 1С основан на русском языке). Язык высокого уровня практически не привязан ни к конкретному процессору, ни к операционной системе (если не используются специфические директивы).
Язык ассемблера – это низкоуровневый язык программирования, на котором вы пишите свои программы. Для каждого процессора существует свой язык ассемблера.
Ассемблер – это специальная программа, которая преобразует (компилирует) исходные тексты вашей программы, написанной на языке ассемблера, в исполняемый файл (файл с расширением EXE или COM). Если быть точным, то для создания исполняемого файла требуются дополнительные программы, а не только ассемблер. Но об этом позже…
В большинстве случаев говорят «ассемблер», а подразумевают «язык ассемблера». Теперь вы знаете, что это разные вещи и так говорить не совсем правильно. Хотя все программисты вас поймут.
ВАЖНО!
В отличие от языков высокого уровня, таких, как Паскаль, Бейсик и т.п., для КАЖДОГО АССЕМБЛЕРА существует СВОЙ ЯЗЫК АССЕМБЛЕРА. Это правило в корне отличает язык ассемблера от языков высокого уровня. Исходные тексты программы (или просто «исходники»), написанной на языке высокого уровня, вы в большинстве случаев можете откомпилировать разными компиляторами для разных процессоров и разных операционных систем. С ассемблерными исходниками это сделать будет намного сложнее. Конечно, эта разница почти не ощутима для разных ассемблеров, которые предназначены для одинаковых процессоров. Но в том то и дело, что для КАЖДОГО ПРОЦЕССОРА существует СВОЙ АССЕМБЛЕР и СВОЙ ЯЗЫК АССЕМБЛЕРА. В этом смысле программировать на языках высокого уровня гораздо проще. Однако за все удовольствия надо платить. В случае с языками высокого уровня мы можем столкнуться с такими вещами как больший размер исполняемого файла, худшее быстродействие и т.п.
В этой книге мы будем говорить только о программировании для компьютеров с процессорами Intel (или совместимыми). Для того чтобы на практике проверить приведённые в книге примеры, вам потребуются следующие программы (или хотя бы некоторые из них):
Emu8086. Хорошая программа, особенно для новичков. Включает в себя редактор исходного кода и некоторые другие полезные вещи. Работает в Windows, хотя программы пишутся под DOS. К сожалению, программа стоит денег (но оно того стоит))).
TASM – Турбо Ассемблер от фирмы Borland. Можно создавать программы как для DOS так и для Windows. Тоже стоит денег и в данный момент уже не поддерживается (да и фирмы Borland уже не существует). А вообще вещь хорошая.
MASM – Ассемблер от компании Microsoft (расшифровывается как МАКРО ассемблер, а не MicrosoftAssembler, как думают многие непосвящённые). Пожалуй, самый популярный ассемблер для процессоров Intel. Поддерживается до сих пор. Условно бесплатная

программа. То есть, если вы будете покупать её отдельно, то она будет стоить денег. Но она доступна бесплатно подписчикам MSDN и входит в пакет программ VisualStudio от Microsoft.
WASM – ассемблер от компании Watcom. Как и все другие, обладает преимуществами и недостатками.
Debug - обладает скромными возможностями, но имеет большой плюс - входит в стандартный набор Windows. Поищите ее в папке WINDOWS\COMMAND или WINDOWS\SYSTEM32. Если не найдете, тогда в других папках каталога WINDOWS.
Желательно также иметь какой-нибудь шестнадцатеричный редактор. Не помешает и досовский файловый менеджер, например Волков Коммандер (VC) или Нортон Коммандер (NC). С их помощью можно также посмотреть шестнадцатеричные коды файла, но редактировать нельзя. Бесплатных шестнадцатеричных редакторов в Интернете довольно много. Вот один из них: McAfeeFileInsight v2.1. Этот же редактор можно использовать для работы с исходными текстами программ. Однако мне больше нравится делать это с помощью следующего редактора:
Текстовый редактор. Необходим для написания исходных текстов ваших программ. Могу порекомендовать бесплатный редактор PSPad, который поддерживает множество языков программирования, в том числе и язык Ассемблера.
Все представленные в этой книге программы (и примеры программ) проверены на работоспособность. И именно эти программы используются для реализации примеров программ, приведённых в данной книге.
И еще – исходный код, написанный, например для Emu8086, будет немного отличаться от кода, написанного, например, для TASM. Эти отличия будут оговорены.
Большая часть программ, приведённых в книге, написана для MASM. Во-первых, потому что этот ассемблер наиболее популярен и до сих пор поддерживается. Во-вторых, потому что он поставляется с MSDN и с пакетом программ VisualStudio от Microsoft. Ну и в третьих, потому что я являюсь счастливым обладателем лицензионной копии MASM.
Если же у вас уже есть какой-либо ассемблер, не вошедший в перечисленный выше список, то вам придётся самостоятельно разобраться с его синтаксисом и почитать руководство пользователя, чтобы научиться правильно с ним работать. Но общие рекомендации, приведённые в данной книге, будут справедливы для любых (ну или почти для любых) ассемблеров.

2)Язык Ассемблера. Основные команды.

Язык ассемблера — тип языка программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком. Часто для краткости его называют просто ассемблером, что не верно.
Содержание языка
Команды языка ассемблера один в один соответствуют командам процессора и, фактически, представляют собой удобную символьную форму записи (мнемокод) команд и их аргументов. Также язык ассемблера обеспечивает базовые программные абстракции: связывание частей программы и данных через метки с символьными именами (при ассемблировании для каждой метки высчитывается адрес, после чего каждое вхождение метки заменяется на этот адрес) и директивы.
Директивы ассемблера позволяют включать в программу блоки данных (описанные явно или считанные из файла); повторить определённый фрагмент указанное число раз; компилировать фрагмент по условию; задавать адрес исполнения фрагмента, отличный от адреса расположения в памяти[уточнить!]; менять значения меток в процессе компиляции; использовать макроопределения с параметрами и др.
Каждая модель процессора, в принципе, имеет свой набор команд и соответствующий ему язык (или диалект) ассемблера.

Достоинства и недостатки

Достоинства языка ассемблера

Минимальное количество избыточного кода, то есть использование меньшего количества команд и обращений в память, позволяет увеличить скорость и уменьшить размер программы.

Обеспечение полной совместимости и максимального использования возможностей нужной платформы: использование специальных инструкций и технических особенностей данной платформы.

При программировании на ассемблере становятся доступными специальные возможности: непосредственный доступ к аппаратуре, портам ввода-вывода и особым регистрам процессора, а также возможность написания самомодифицирующегося кода (то есть метапрограммирование, причём без необходимости программного интерпретатора). Последние технологии безопасности, внедряемые в операционные системы, не позволяют делать самомодифицирующегося кода, так как исключают одновременную возможность исполнения инструкций и запись в одном и том же участке памяти (технология W^X в BSD-системах, DEP в Windows).

Недостатки языка ассемблера

Большие объёмы кода и большое число дополнительных мелких задач, что приводит к тому, что код становится очень сложно читать и понимать, а следовательно усложняется отладка и доработка программы, а также трудность реализации парадигм программирования и любых других соглашений. что приводит к сложности совместной разработки.

Меньшее количество доступных библиотек, их малая совместимость между собой.

Непереносимость на другие платформы (кроме двоично совместимых).

Применение

Напрямую вытекает из достоинств и недостатков.

Поскольку большие программы на ассемблере писать крайне неудобно, их пишут на языках высокого уровня. На ассемблере же пишут небольшие фрагменты или модули, для которых критически важны:

быстродействие (драйверы);

размер кода (загрузочные сектора, программное обеспечение для микроконтроллеров и процессоров с ограниченными ресурсами, вирусы, программные защиты);

специальные возможности: работа напрямую с аппаратурой или машинным кодом, то есть загрузчики операционных систем, драйверы, вирусы, системы защиты.

Связывание ассемблерного кода с другими языками

Поскольку на ассемблере чаще всего пишут лишь фрагменты программы, их необходимо связывать с остальными частями на других языках. Это достигается 2 основными способами:

На этапе компиляции — вставка в программу ассемблерных фрагментов (англ. inlineassembler) специальными директивами языка, в том числе написание процедур на языке ассемблера. Способ удобен для несложных преобразований данных, но полноценного ассемблерного кода, с данными и подпрограммами, включая подпрограммы с множеством входов и выходов, не поддерживаемых высокоуровневыми языками, с помощью него сделать нельзя.

На этапе компоновки, или раздельная компиляция. Для взаимодействия скомпонованных модулей достаточно, чтобы связующие функции[3] поддерживали нужные соглашения о вызовах (англ. callingconventions) и типы данных. Написаны же отдельные модули могут быть на любых языках, в том числе и на ассемблере.

Синтаксис

Общепринятого стандарта для синтаксиса языков ассемблера не существует. Однако, существуют стандарты, которых придерживаются большинство разработчиков языков ассемблера. Основными такими стандартами являются Intel-синтаксис и AT&T-синтаксис.

Инструкции

Общий формат записи инструкций одинаков для обоих стандартов:

[метка:] опкод [операнды] [;комментарий]

где опкод — непосредственно мнемоника инструкции процессору. К ней могут быть добавлены префиксы (повторения, изменения типа адресации и пр.).

В качестве операндов могут выступать константы, названия регистров, адреса в оперативной памяти и пр.. Различия между стандартами Intel и AT&T касаются, в основном, порядка перечисления операндов и их синтаксиса при различных методах адресации.

Используемые мнемоники обычно одинаковы для всех процессоров одной архитектуры или семейства архитектур (среди широко известных - мнемоники процессоров и контроллеров Motorola, ARM, x86). Они описываются в спецификации процессоров.

Возможные исключения:

Если ассемблер использует кроссплатформенный AT&T-синтаксис (оригинальные мнемоники приводятся к синтаксису AT&T)

Если изначально существовало два стандарта записи мнемоник (система команд была наследована от процессора другого производителя).

Например, процессор Zilog Z80 наследовал систему команд Intel i8080, расширил ее и поменял мнемоники (и обозначения регистров) на свой лад. Например сменил интеловские mov[4] на ld. Процессоры MotorolaFireball наследовали систему команд Z80, несколько её урезав. Вместе с тем, Motorola официально вернулась к мнемоникам Intel. И в данный момент половина ассемблеров для Fireball работает с интеловскими мнемониками, а половина с мнемониками Zilog.

Директивы

Кроме инструкций, программа может содержать директивы: команды, не переводящиеся непосредственно в машинные инструкции, а управляющие работой компилятора. Набор и синтаксис их значительно разнятся и зависят не от аппаратной платформы, а от используемого компилятора (порождая диалекты языков в пределах одного семейства архитектур). В качестве "джентельменского набора" директив можно выделить:

определение данных (констант и переменных)

управление организацией программы в памяти и параметрами выходного файла

задание режима работы компилятора

всевозможные абстракции (т.е. элементы языков высокого уровня) - от оформления процедур и функций (для упрощения реализации парадигмы процедурного программирования) до условных конструкций и циклов (для парадигмы структурного программирования)

макросы

Пример программы

Пример программы Helloworld для MS-DOS для архитертуры x86 на диалекте TASM:

```
.MODEL TINY
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
ORG 100h
START:
mov ah,9
movdx,OFFSETMsg
int 21h
int 20h
Msg DB 'Hello World',13,10,'$'
```

CODE ENDS
END START

1. 5 Лекция №9-11 (6 часов).
Тема: «Структуры запоминающих устройств ЭВМ. Структура ОЗУ»

1.5.1 Вопросы лекции:

1) Загрузочный сектор.
2) Файловая система FAT.

1.5.2 Краткое содержание вопросов:

1) Загрузочный сектор.

Загрузочный сектор, бутсектор (англ. bootsector, Volumebootsector (Volumebootrecord), Partitionbootsector) — это особый сектор на жёстком диске,дискете или другом дисковом устройстве хранения информации. (Для дискеты это первый физический сектор, для жёсткого диска — первый физический сектор для каждого раздела.) В процессе загрузки компьютера с дискеты он загружается в память программой POST (в компьютерах архитектуры IBM PC обычно с адреса 0000:7c00), ему передается управление командой longjump.
Загрузочный сектор, иногда называемый stage1 (то есть первым этапом загрузки операционной системы), загружает программу второго этапа загрузки операционной системы stage2 (вторичный загрузчик, иногда в качестве stage2 загружается bootmanager или программа авторизации и защиты доступа). В некоторых ОС роль stage1 выполняет MBR, и при загрузке ОС с жёсткого диска загрузочный сектор не используется.

2)Файловая система FAT.
Файловая система FAT (FileAllocationTable) была разработана Биллом Гейтсом и Марком Макдональдом в 1977 году.
Сейчас существуют три типа файловой системы FAT:
FAT12 – поддерживает очень небольшие объемы дисков, поэтому сейчас она применяется только на дискетах.
FAT16 – используется на винчестерах и поддерживает диски объемом до 2 Гб, поэтому сейчас данная файловая система практически не используется.
FAT32 – теоретически поддерживаются диски объемом до 2 Тб. Поддерживается, начиная с операционной системы Windows 95 OSR2. Данная файловая система сейчас достаточно популярна, хотя в последние годы многие пользователи Windows XP предпочитают использовать NTFS (NewTechnologyFileSystem).
Структура

Загрузочный сектор	Таблица FAT (1-я копия)	Таблица FAT (2-я копия)	Корневой каталог	Область данных

Рис. 1. Структура файловой системы FAT
– Загрузочный сектор
В начале раздела диска с файловой системой FAT располагается загрузочный сектор. Он необходим для начальной загрузки компьютера. Так же в нем располагается информация о параметрах данного раздела.
– Таблицаразмещенияфайлов (File Allocation Table)
Вся область данных диска разделена на кластеры – блоки, размер которых задается при форматировании диска. На дискете, например, размер кластера равен 512-ти байтам. А на современных винчестерах с объемом диска более 32 Гб размер кластера равен 32 Кб. Каждый файл и каталог занимает один или несколько кластеров. Таким образом, образуются цепочки кластеров.

В таблице размещения файлов каждый кластер помечается специальным образом. Размер метки в битах для каждого кластера указывается в названии файловой системы. Т.е. для файловой системы FAT16 размер метки будет равен 16-ти байтам, для FAT32 – 32-м и т.д. Всего существует три типа меток для кластеров:
Свободный кластер – кластер, в который будут записываться новые файлы и каталоги.
Занятый кластер – в метке указывается следующий кластер в цепочке. Если цепочка кластеров заканчивается, то кластер помечается особой меткой.
BAD-блок – кластер с ошибками доступа. Помечается при форматировании диска, что бы исключить в последующем доступ к нему.
Повреждение таблицы размещения файлов полностью уничтожает структуру файловой системы, поэтому на диске всегда хранится две копии таблицы.
– Корневой каталог
Область диска, в котором располагается информация о корневом каталоге. Размер ее ограничен, поэтому в корневом каталоге диска может находиться не более 512-ти файлов и подкаталогов.
– Область данных
Оставшаяся часть раздела, на которой размещается содержимое файлов и каталогов.
FAT32 и NTFS
Сейчас основным конкурентом популярной файловой системы FAT32 является NTFS. Файловая система NTFS получила популярность с распространением операционной системы Windows XP.
– FAT32
Основным преимуществом этой файловой системы является ее простота и совместимость со старыми операционными системами. Для этой файловой системы существует большое количество подробной документации. Существует ограничение на максимальный размер файла – 4 Гб. Сбои в системе часто приводят к повреждением одного или нескольких файлов. Однако, при серьезных повреждениях, восстановить информацию гораздо проще, чем в случае с NTFS.
– NTFS
Основными преимуществами файловой системы NTFS являются ее защищенность от несанкционированного доступа. В этой файловой системе отсутствуют ограничения на размер файлов и каталогов. Так же ее особенность является журналирование – запись всех операций перед их выполнением в специальный журнал. В случае, если во время выполнения операций с файловой системой произойдет сбой (зависание операционной системы, отключение электричества и т.п.), то она на основе записей в журнале сможет вернуть себя в прежнее состояние. Однако, в случае серьезного сбоя восстановить информацию будет очень сложно, подчас невозможно. Причиной тому является отсутствие официальной документации файловой системы от Microsoft. Так же недостатком NTFS является несовместимость со старыми версиями операционных систем (Windows 95, 98).

1. 6 Лекция №12-13 (4 часа).

Тема: «Устройства хранения данных. Структура основной памяти**»**

1.6.1 Вопросы лекции:

1) Флэш-носители.
2) Дисковая память.

1.6.2 Краткое содержание вопросов:

1) Флэш-носители.

Флеш-память (англ. *flashmemory*) — разновидность полупроводниковой технологии электрически перепрограммируемой памяти (EEPROM). Это же слово используется в электронной схемотехнике для обозначения технологически законченных решений постоянных запоминающих устройств в виде микросхем на базе этой полупроводниковой технологии. В быту это словосочетание закрепилось за широким классом твердотельных устройств хранения информации.

Благодаря компактности, дешевизне, механической прочности, большому объёму, скорости работы и низкому энергопотреблению, флеш-память широко используется в цифровых портативных устройствах и носителях информации. Серьёзным недостатком данной технологии является ограниченный срок эксплуатации носителей,[1][2] а также чувствительность к электростатическому разряду.

2) Дисковая память.

Носителями информации являются поверхности гибких и жестких дисков, в качестве немагнитных основ которых используются соответственно майлар (как и в магнитных лентах) и алюминиевые (в ряде случаев стеклянные) круги (диски). Стеклянные диски являются менее критичными к температурным изменениям и позволяют увеличить плотность записи информации. В настоящее время наиболее широкое распространение получили диски с напыленным магнитным слоем, а точнее, с металлической пленкой (например, кобальт).

Перед осуществлением записи на магнитный диск он должен быть специальным образом инициализирован – отформатирован. В результате форматирования на поверхности образуются концентрические окружности (синхронизирующие метки диска), называемые **дорожками** (track). Количество дорожек зависит от типа диска. Дорожки разбиваются на участки фиксированной длины, называемые **секторами.** Количество секторов на дорожке определяется типом и **форматом** диска, и они в основном одинаковы для всех дорожек. IBM PC-совместимые ПК могут работать с несколькими размерами секторов от 128 до 1024 байт. Стандартным сектором считается сектор из 512 байт. Данные любого размера (разрядности) размещаются в секторах с фиксированным размером, а дисковые операции записи и считывания производятся с целыми секторами.

Дорожки и сектора нумеруются с нуля, начиная с внешнего края диска, при этом сектор с нулевым номером на каждой дорожке резервируется для системных целей. Диски имеют две стороны. Так как накопители на жестких дисках могут состоять из нескольких дисков (стопка), то совокупность всех дорожек, по одной на каждой стороне с одинаковыми номерами, образует **цилиндр** с номером соответствующей дорожки.

1. 7 Лекция №14-15 (4 часа).
Тема: «Устройства хранения данных»

1.7.1 Вопросы лекции:

1) Интерфейсы COM
2) Интерфейсы USB

1.7.2 Краткое содержание вопросов:

1)Интерфейсы COM

- (от COM — англ. ComponentObjectModel и англ. interface — взаимодействие) — набор абстрактных функций и свойств, через

который программы взаимодействует с COM-компонентом. Состав этого набора объявляется независимо от компонента, и публикуется, как правило, на языке IDL. Компонент реализует работу в соответствии с декларируемыми интерфейсами. В разных языках программирования для описания спецификации интерфейса предусмотрены различные средства. К COM-интерфейсам стандарты предъявляют жёсткие требования по реализации определённых функций, поэтому иногда образно говорят, что интерфейс — это контракт, который обязуется выполнять компонент. С конца XX века многие среды программирования начали внедрять у себя понятие интерфейса для поддержки технологии COM или сходных технологий.
Интерфейс, в отличие от класса, содержит только описание методов, без их реализации.

2)Интерфейсы USB
USB (англ.UniversalSerialBus) - универсальная последовательная шина, предназначенная для подключения периферийных устройств. Шина USB представляет собой последовательный интерфейс передачи данных для высоко-, средне- и низкоскоростных периферийных устройств.
Разработка спецификаций на шину USB производится в рамках международной некоммерческой организации USB ImplementersForum (USB-IF), объединяющей разработчиков и производителей оборудования с шиной USB.
Для подключения периферийных устройств к шине USB используется четырёхпроводный кабель, при этом два провода (витая пара) в дифференциальном включении используются для приёма и передачи данных, а два провода - для питания периферийного устройства. Благодаря встроенным линиям питания, USB позволяет подключать периферийные устройства без собственного источника питания (максимальная сила тока, потребляемого устройством по линиям питания шины USB, не должна превышать 500 мА).
К одному контроллеру шины USB можно подсоединить до 127 устройств по топологии "звезда", в том числе и концентраторы, к которым можно еще присоединить 127 устройств.
В настоящее время широко используются устройства, выполненные в соответствии со спецификацией USB 2.0. Ведётся внедрение в производство устройств спецификации USB 3.0.

1. 8 Лекция №16-17 (4 часа).
Тема: «Аудиосистема ПК. Коммуникационные устройства»

1.8.1 Вопросы лекции:

1) Сервера для рабочей группы.
2) Многопроцессорные комплексы.

1.8.2 Краткое содержание вопросов:

1) Сервера для рабочей группы.

Давайте же узнаем подробнее, что это такое и что он собой представляет.

Сервер – это специализированное аппаратное устройство, которое выполняет те или иные задачи, как удалённо (что бывает чаще всего), так и локально на месте. Теперь расскажем об этом более понятным языком. Итак, сервер – это тот же самый компьютер, только в большинстве случаев мощнее. Сервер, как и компьютер, состоит из: процессора, материнской платы, оперативной памяти и жёсткого диска. Чаще всего для крупных серверов используют специально предназначенные для этого комплектующие, но, тем не менее, есть сервера из тех комплектующих, которые используют и простые пользователи.

Раньше сервера были в обычном системном блоке (корпусе), но это очень некомпактно и неудобно, так как в большинстве организаций имеется необходимости в нескольких серверах, и такие «ящики» были очень непрактичными. Благодаря новым технологиям, начиная с конца 2000-ых годов, сервера стали иметь несколько другую форму. Теперь они находятся в специальном отсеке, который монтируется в стойку. Данные стойки имеют форму шкафов, отсюда и получили такое разговорное название у компьютерщиков.

Управление большим количеством серверов, которые находятся в стойках, производится удалённо. То есть инженер, находясь как в соседнем кабинете, так и в другом городе – может на расстоянии управлять серверами. Установка же операционных системы на сервер производится на месте.

Сервер управляется, как было сказано только что, операционной системой: как Linux, так иWindows. Существует достаточно много версий операционных систем предназначенных для управления серверов, но стоит отметить, что это специальные их версии. То есть, для сервера устанавливают, например не Windows 7 или 8, а допустим WindowsServer 2012. То же самое касается и Linux: на сервер устанавливается, например не LinuxUbuntu, а LinuxUbuntu 12.04 Server. Если у Windows только одна серверная операционная система, то у Linux их несколько. Например, есть ещё серверные версии Debian и CentOS.

3) Многопроцессорные комплексы.

Появление первых моделей семейства IBM/360 связано с бурным ростом использования ЭВМ в научно-технических расчетах. При этом важнейшую роль отводили системам машинного контроля безошибочной работы аппаратуры. Для контроля использовались как аппаратные средства (контроль по четности и другие корректирующие коды и т.д.), так и программные средства (двойной просчет). Естественным стало и двойное параллельное выполнение программ на многомашинных комплексах со сравнением промежуточных результатов. В этих комплексах каждая ЭВМ использовала свою операционную систему, а одна из них – дополнительно программу организации совместной работы ЭВМ в комплексе.

Схема двухмашинного комплекса на основе моделей IBM/360 представлена на рис. 1.4.

Каждая модель комплекса содержит процессор, модули памяти и систему ввода/вывода (каналы в/в). Обмен данными производится через систему ввода/вывода. На рис. 1.4 для связи между моделями комплекса использовано специальное устройство – адаптер «канал-канал».

Это однобайтный регистр передаваемых данных и два контроллера со стороны интерфейсов ввода/вывода, соединяемых моделей комплекса.

Передача данных через адаптер «канал-канал» начинается по команде программы ввода/вывода со стороны одного из процессоров. Этой командой может быть команда "записать". Контроллер адаптера по этой команде посылает сигнал прерывания в процессор смежной ЭВМ. В программе прерывания предусмотрена команда обращения к этому же адаптеру, но уже по чтению. Адаптер, получив задания от процессоров, отвечает положительными квитанциями. Так организуется передача данных. Конец передачи данных определен счетчиками данных в командах канала.

По окончании передачи данных контроллеры адаптеров «канал-канал» посылают каналам байты состояния. Этим заканчивается выполнение команды канала. Передача данных прекращается по окончании всех команд программы ввода/вывода. Программа канала

может завершиться штатно или по сбою. В любом случае контроллеры адаптеров «канал-канал» посылают процессорам сигналы прерывания для анализа условий окончания программы ввода/вывода.

Интерфейс прямого управления предназначен для передачи управляющих сигналов между процессорами.

Для увеличения производительности и более эффективного использования большого парка периферийных устройств использовались и многопроцессорные системы, например – двухпроцессорные.

Многомашинные комплексы в настоящее время определяются как асимметричные, слабосвязанные многопроцессорные системы; системы с неоднородным доступом к памяти (NUMA–Non-UniformMemoryAccess).

Многопроцессорные системы

Многопроцессорные системы определяются как симметричные мультипроцессорные системы (SMP системы – SummitriesMultiprocessor). Все процессоры системы работают в едином виртуальном и физическом пространстве оперативной памяти. Любой из процессоров может обслуживать внешние прерывания. Это сильно связанные системы.

1. 9 Лекция №18-21 (8 часов).
Тема: «Принципы построения процессора. Структура машинных команд и способы адресации**»**

1.9.1 Вопросы лекции:

1) История возникновения и развития вычислительной техники
2) Классификация компьютеров. Поколения вычислительной техники

1.9.2 Краткое содержание вопросов:

1) История возникновения и развития вычислительной техники

Первым устройством, предназначенным для облегчения счета, были счеты. С помощью костяшек счетов можно было совершать операции сложения и вычитания и несложные умножения.

1642 г. — французский математик Блез Паскаль сконструировал первую механическую счетную машину «Паскалина», которая могла механически выполнять сложение чисел.

1673 г. — Готфрид Вильгельм Лейбниц сконструировал арифмометр, позволяющий механически выполнять четыре арифметических действия.

Первая половина XIX в. — английский математик Чарльз Бэббидж попытался построить универсальное вычислительное устройство, то есть компьютер. Бэббидж называл его аналитической машиной. Он определил, что компьютер должен содержать память и управляться с помощью программы. Компьютер по Бэббиджу — это механическое устройство, программы для которого задаются посредством перфокарт — карт из плотной бумаги с информацией, наносимой с помощью отверстий (они в то время уже широко употреблялись в ткацких станках).

1941 г. — немецкий инженер Конрад Цузе построил небольшой компьютер на основе нескольких электромеханических реле.

1943 г. — в США на одном из предприятий фирмы IBM ГовардЭйкен создал компьютер под названием «Марк-1». Он позволял проводить вычисления в сотни раз быстрее, чем вручную (с помощью арифмометра), и использовался для военных расчетов. В нем использовалось сочетание электрических сигналов и механических приводов. «Марк-1» имел размеры: 15 * 2—5 м и содержал 750 000 деталей. Машина была способна перемножить два 32-разрядных числа за 4 с.

1943 г. — в США группа специалистов под руководством Джона Мочли и ПроспераЭкерта начала конструировать компьютер ENIAC на основе электронных ламп.

1945 г. — к работе над ENIAC был привлечен математик Джон фон Нейман, который подготовил доклад об этом компьютере. В своем докладе фон Нейман сформулировал общие принципы функционирования компьютеров, т. е. универсальных вычислительных устройств. До сих пор подавляющее большинство компьютеров сделано в соответствии с теми принципами, которые изложил Джон фон Нейман.

1947 г. — Экертом и Мочли начата разработка первой электронной серийной машины UNIVAC (UniversalAutomaticComputer). Первый образец машины (UNIVAC-1) был построен для бюро переписи США и пущен в эксплуатацию весной 1951 г. Синхронная, последовательного действия вычислительная машина UNIVAC-1 была создана на базе ЭВМ ENIAC и EDVAC. Работала она с тактовой частотой 2,25 МГц и содержала около 5000 электронных ламп. Внутреннее запоминающее устройство емкостью 1000 12-разрядных десятичных чисел было выполнено на 100 ртутных линиях задержки.

1949 г. — английским исследователем Морнсом Уилксом построен первый компьютер, в котором были воплощены принципы фон Неймана.

1951 г. — Дж. Форрестер опубликовал статью о применении магнитных сердечников для хранения цифровой информации, В машине «Whirlwind-1» впервые была применена память на магнитных сердечниках. Она представляла собой 2 куба с 32-32-17 сердечниками, которые обеспечивали хранение 2048 слов для 16-разрядных двоичных чисел с одним разрядом контроля на четность.

1952 г. — фирма IBM выпустила свой первый промышленный электронный компьютер IBM 701, который представлял собой синхронную ЭВМ параллельного действия, содержащую 4000 электронных ламп и 12 000 диодов. Усовершенствованный вариант машины IBM 704 отличался высокой скоростью работы, в нем использовались индексные регистры и данные представлялись в форме с плавающей запятой.

После ЭВМ IBM 704 была выпущена машина IBM 709, которая в архитектурном плане приближалась к машинам второго и третьего поколений. В этой машине впервые была применена косвенная адресация и впервые появились каналы ввода — вывода.

1952 г. — фирма RemingtonRand выпустила ЭВМ UNIVAC-t 103, в которой впервые были применены программные прерывания. Сотрудники фирмы RemingtonRand

использовали алгебраическую форму записи алгоритмов под названием «ShortCode» (первый интерпретатор, созданный в 1949 г. Джоном Мочли).

1956 г. — фирмой IBM были разработаны плавающие магнитные головки на воздушной подушке. Изобретение их позволило создать новый тип памяти — дисковые запоминающие устройства (ЗУ), значимость которых была в полной мере оценена в последующие десятилетия развития вычислительной техники. Первые ЗУ на дисках появились в машинах IBM 305 и RAMAC. Последняя имела пакет, состоявший из 50 металлических дисков с магнитным покрытием, которые вращались со скоростью 12000 об. /мин. На поверхности диска размещалось 100 дорожек для записи данных, по 10 000 знаков каждая.

1956 г. — фирма Ferranti выпустила ЭВМ «Pegasus», в которой впервые нашла воплощение концепция регистров общего назначения (РОН). С появлением РОН устранено различие между индексными регистрами и аккумуляторами, и в распоряжении программиста оказался не один, а несколько регистров-аккумуляторов.

1957 г. — группа под руководством Д. Бэкуса завершила работу над первым языком программирования высокого уровня, получившим название ФОРТРАН. Язык, реализованный впервые на ЭВМ IBM 704, способствовал расширению сферы применения компьютеров.

1960-е гг. — 2-е поколение ЭВМ, логические элементы ЭВМ реализовываются на базе полупроводниковых приборов-транзисторов, развиваются алгоритмические языки программирования, такие как Алгол, Паскаль и другие.

1970-е гг. — 3-е поколение ЭВМ, интегральные микросхемы, содержащие на одной полупроводниковой пластине тысячи транзисторов. Начали создаваться ОС, языки структурного программирования.

1974 г. — несколько фирм объявили о создании на основе микропроцессора Intel-8008 персонального компьютера — устройства, выполняющего те же функции, что и большой компьютер, но рассчитанного на одного пользователя.

1975 г. — появился первый коммерчески распространяемый персональный компьютер Альтаир-8800 на основе микропроцессора Intel-8080. Этот компьютер имел оперативную память всего 256 байт, клавиатура и экран отсутствовали.

Конец 1975 г. — Пол Аллен и Билл Гейтс (будущие основатели фирмы Microsoft) создали для компьютера «Альтаир» интерпретатор языка Basic, позволивший пользователям просто общаться с компьютером и легко писать для него программы.

Август 1981 г. — компания IBM представила персональный компьютер IBM PC. В качестве основного микропроцессора компьютера использовался 16-разрядный микропроцессор Intel-8088, который позволял работать с 1 мегабайтом памяти.

1980-е гг. — 4-е поколение ЭВМ, построенное на больших интегральных схемах. Микропроцессоры реализовываются в виде единой микросхемы, Массовое производство персональных компьютеров.

1990-е гг. — 5-е поколение ЭВМ, сверхбольшие интегральные схемы. Процессоры содержат миллионы транзисторов. Появление глобальных компьютерных сетей массового пользования.

2000-е гг. — 6-е поколение ЭВМ. Интеграция ЭВМ и бытовой техники, встраиваемые компьютеры, развитие сетевых вычислений.

2) Классификация компьютеров. Поколения вычислительной техники

Компьютеры могут быть классифицированы по разным признакам, например по габаритам, по областям применения, по быстродействию, по функциям, по этапам создания и еще по многим другим параметрам.

Мы рассмотрим классификацию по обобщенному параметру, где в разной степени учтено несколько характерных признаков:

назначение и роль компьютеров в системе обработки информации;

условия взаимодействия человека и компьютера;

габариты компьютера;

ресурсные возможности компьютера.

В соответствии с вышесформулированными признаками и тенденциями развития компьютерной техники можно рассмотреть следующую классификацию компьютеров:

портативные компьютеры;

микрокомпьютеры, в том числе — персональные компьютеры;

мэйнфреймы (универсальные компьютеры);

суперкомпьютеры.

Портативные компьютеры обычно нужны руководителям предприятий, менеджерам, учёным, журналистам, которым приходится работать вне офиса — дома, на презентациях или во время командировок.

Основные разновидности портативных компьютеров:

Микрокомпьютеры — это компьютеры, в которых центральный процессор выполнен в виде микропроцессора.

Продвинутые модели микрокомпьютеров имеют несколько микропроцессоров. Производительность компьютера определяется не только характеристиками применяемого микропроцессора, но и ёмкостью оперативной памяти, типами периферийных устройств, качеством конструктивных решений и др.

Микрокомпьютеры представляют собой инструменты для решения разнообразных сложных задач. Их микропроцессоры с каждым годом увеличивают мощность, а периферийные устройства — эффективность. Быстродействие — порядка 1 - 10 миллионов операций в сек.

Разновидность микрокомпьютера — микроконтроллер. Это основанное на микропроцессоре специализированное устройство, встраиваемое в систему управления или технологическую линию.

В класс персональных компьютеров входят различные машины — от дешёвых домашних и игровых с небольшой оперативной памятью, с памятью программы на кассетной ленте и обычным телевизором в качестве дисплея, до сверхсложных машин с мощным процессором, винчестерским накопителем ёмкостью в десятки Гигабайт, с цветными графическими устройствами высокого разрешения, средствами мультимедиа и другими дополнительными устройствами.

Персональныйкомпьютер должен удовлетворять следующим требованиям:

стоимость от нескольких сотен до 5-10 тысяч долларов;

наличие внешних ЗУ на магнитных дисках;

объём оперативной памяти не менее 4 Мбайт;

наличие операционной системы;

способность работать с программами на языках высокого уровня;
ориентация на пользователя-непрофессионала (в простых моделях).

Мейнфрейм - это синоним понятия "большая универсальная ЭВМ". Мейнфреймы и до сегодняшнего дня остаются наиболее мощными (не считая суперкомпьютеров)вычислительными системами общего назначения, обеспечивающими непрерывный круглосуточный режим эксплуатации. Они могут включать один или несколько процессоров, каждый из которых, в свою очередь, может оснащаться векторными сопроцессорами (ускорителями операций с суперкомпьютерной производительностью). В нашем сознании мейнфреймы все еще ассоциируются с большими по габаритам машинами, требующими специально оборудованных помещений с системами водяного охлаждения и кондиционирования. Однако это не совсем так. Прогресс в области элементно-конструкторской базы позволил существенно сократить габариты основных устройств. Наряду со сверхмощными мейнфреймами, требующими организации двухконтурной водяной системы охлаждения, имеются менее мощные модели, для охлаждения которых достаточно принудительной воздушной вентиляции, и модели, построенные по блочно-модульному принципу и не требующие специальных помещений и кондиционеров.

Основными поставщиками мейнфреймов являются известные компьютерные компании IBM, Amdahl, ICL, SiemensNixdorf и некоторые другие, но ведущая роль принадлежит безусловно компании IBM. Именно архитектура системы IBM/360, выпущенной в 1964 году, и ее последующие поколения стали образцом для подражания. В нашей стране в течение многих лет выпускались машины ряда ЕС ЭВМ, являвшиеся отечественным аналогом этой системы. В архитектурном плане мейнфреймы представляют собоймногопроцессорные системы, содержащие один или несколько центральных и периферийных процессоров с общей памятью, связанных между собой высокоскоростными магистралями передачи данных. При этом основная вычислительная нагрузка ложится на центральные процессоры, а периферийные процессоры (в терминологии IBM - селекторные, блок-мультиплексные, мультиплексные каналы и процессоры телеобработки) обеспечивают работу с широкой номенклатурой периферийных устройств.

Они предназначены для решения широкого класса научно-технических задач и являются сложными и дорогими машинами. Их целесообразно применять в больших системах при наличии не менее 200 - 300 рабочих мест.

Централизованная обработка данных на мэйнфрейме обходится примерно в 5 - 6 раз дешевле, чем распределённая обработка при клиент-серверном подходе.

Известный мейнфрейм S/390 фирмы IBM обычно оснащается не менее чем тремя процессорами. Максимальный объём оперативного хранения достигает 342 Терабайт. Производительность его процессоров, пропускная способность каналов, объём оперативного хранения позволяют наращивать число рабочих мест в диапазоне от 20 до 200000 с помощью простого добавления процессорных плат, модулей оперативной памяти и дисковых накопителей.

Десятки мейнфреймов могут работать совместно под управлением одной операционной системы над выполнением единой задачи.

Отличительной особенностью суперкомпьютеров являются векторные процессоры, оснащенные аппаратурой для параллельного выполнения операций с многомерными цифровыми объектами — векторами и матрицами. В них встроены векторные регистры и параллельный конвейерный механизм обработки. Если на обычном процессоре программист выполняет операции над каждым компонентом вектора по очереди, то на векторном — выдаёт сразу векторые команды.

Векторная аппаратура очень дорога, в частности, потому, что требуется много сверхбыстродействующей памяти под векторные регистры.

Наряду с векторно-конвейерной системой обработки данных существует и скалярная система, основанная на выполнении обычных арифметических операций над отдельными числами или парами чисел.
Строго говоря, системы, использующие скалярную обработку данных, по своей производительности уступают суперЭВМ, но у них наблюдаются тенденции, характерные для высокопроизводительных вычислительных систем: необходимость распараллеливания больших задач между процессорами.
Наиболее распространённые суперкомпьютеры — массово-параллельные компьютерные системы. Они имеют десятки тысяч процессоров, взаимодействующих через сложную, иерархически организованую систему памяти.

Супер-компьютеры используются для решения сложных и больших научных задач (метеорология, гидродинамика и т. п.), в управлении, разведке, в качестве централизованных хранилищ информации и т.д.
Элементная база — микросхемы сверхвысокой степени интеграции.
К суперкомпьютерам часто относят и серверы.

Сервер-мощный компьютер в вычислительных сетях, который обеспечивает обслуживание подключенных к нему компьютеров и выход в другие сети.
В зависимости от назначения определяют такие типы серверов:
Сервер приложений обрабатывает запросы от всех станций вычислительной сети и предоставляет им доступ к общим системным ресурсам (базам данных, библиотекам программ, принткрам, факсам и др.).
Файл-сервер-для работы с базами данных и использования файлов информации, хранящихся в ней.
Архивационный сервер-для резервного копирования информации в крупных многосервисных сетях. Он использует накопители на магнитной ленте(стриммеры) со сменными картриджами емкостью до 5 Гбайт. Обычно выполняет ежедневное автоматическое архивирование информации от подключенных серверов и рабочих станций.
Факс-сервер-для организации эффективной многоадресной факсимильной связи, с несколькими факсмодемными платами, со специальной защитой информации от несанкционированного доступа в процессе передачи, с системой хранения электронных факсов.
Почтовый сервер-то же, что и факс-сервер, но для организации электронной почты, с электронными почтовыми ящиками.
Сервер печати-для эффективного использования системных принтеров.
Сервер- телеконференций-компьютер, имеющий программу обслуживания пользователей телеконференциями и новостями, он также может иметь систему автоматической обработки видеоизображений и др.
Любой компьютер, если установить на нем соответствующее сетевое программное обеспечение, способен стать сервером. Кроме того, один компьютер одновременно может выполнять несколько функций-быть, к примеру, почтовым сервером, сервером новостей, сервером приложений и т.д.
Существуют различные классификации компьютерной техники:
по этапам развития (по поколениям);
по архитектуре;
по производительности;
по условиям эксплуатации;
по количеству процессоров;
по потребительским свойствам и т.д.
Четких границ между классами компьютеров не существует. По мере совершенствования структур и технологии производства, появляются новые классы компьютеров, границы существующих классов существенно изменяются.

1. 10 Лекция №22-25 (8 часов).

Тема: «Современные микропроцессоры. Порядок выполнения машинных команд»

1.10.1 Вопросы лекции:

1) Архитектура и структура компьютера
2) Состав и назначение основных устройств ЭВМ

1.10.2 Краткое содержание вопросов:

1) Организация памяти. Микросхемы памяти

Производительность ПК зависит от типа и размера ОП, а это в свою очередь зависит от набора интегральных схем на материнской плате.
Внешний вид микросхем ОП: пластиковая полоска, на ней расположены кремневые «черепашки» – чипы-микросхемы (то есть используется полупроводниковая технология) и имеются «ножевые» контактные разъемы.
Устройства памяти характеризуются следующими основными показателями:
временем доступа (быстродействием). Время доступа – промежуток времени, за который может быть записано (прочитано) содержимое ячейки памяти.
емкостью (определяет количество ячеек (битов) в устройстве памяти).
стоимостью.
потребляемой мощностью (электропотреблением).
Существует 2 модуля памяти, отличающиеся формой, внутренней архитектурой, скоростью работы:SIMM и DIMM.
I. SIMM (SINGLE IN-LINE MEMORY MODULES) (SRAM)
бывают двух типов (отличающихся количеством контактов).
1. 30-контактные модули SIMM. Бывают 1 и 4 Мб. Практически сегодня исчезли из продажи для компьютеров 386, 286-процессором. Сегодня им нашлось интересное применение – в качестве ОП, устанавливаемой в некоторые звуковые платы, например, GreafiveSoundBlaster 32 (AWE-32) GravisUltraSoundPnP. Однако новая карта AWE-64 уже содержит свои модули ОП, эта память не нужна.
2. 72-контактные SIMM (на 1, 4, 8, 16, 32, 64 Мб, редко 128 Мб). Внешний вид неизменный, а вот тип устанавливаемой на них памяти меняется (тип памяти указывается на микросхеме).
a) самый старый (редко сейчас встречающийся) – FPM DRAM (или просто DRAM – DynamicRandomAccessMemory – динамическая ОП). Работала на 486 и первых Pentium.
b) модифицированный тип EDO DRAM (или EDO – Extendeddataoutput).
Микросхемы SIMM выпускаются одинарной и двойной плотности, с контролем четности и без (использование контроля четности позволяет парировать одиночную ошибку памяти). Модули отличаются и по скорости доступа 60 и 70 наносекунд, чем скорость меньше, тем быстрее доступ. 60 наносекунд быстрее 70 наносекунд. Модули SIMM в материнской плате Pentium и Pentium MMX устанавливаются только попарно, образуя так называемый банк.
Пример необходимо 32 Мб => 2 модуля SIMM по 16 Мб.
необходимо 64 Мб => 4 модуля SIMM по 16 Мб или 2 модуля SIMM по 32Мб.
В рамках одного банка можно использовать только одинаковые по емкости и скорости доступа модули SIMM. Если на вашей материнской плате 4 слота для модулей памяти SIMM, то можно сформировать два банка различной емкости.

II. DIMM (SDRAM DUAL IN-LINE MEMORY MODULES).
Появился впервые у MMX- компьютеров, стал основой для PII., поэтому у PII редко бывают SIMM-разъемы. DIMM не обязательно должно быть четное число. Модули DIMM бывают емкостью 16, 32, 64, 128, 256, 512 Мб
Виды DIMM.

EDO SD RAM (Synchronous DRAM) – синхронизируемая динамическая ОП)
SD RAM (SINGLE DATA RATE RANDOM ACCESS MEMORY).ЗУПВ с одинарной скоростью передачи данных, которая в зависимости от тактовой частоты называется памятью PC100 и PC133. Микросхема на 168 контактов, является сегодня самой «медленной» из семейства DIMM-модулей памяти, Время доступа = 10-20 наносекунд. Верхний предел ее тактовой частоты 133 МГц. И все же этот тип ОП вполне подходит для большинства офисных и
домашних ПК. Пропускная способность 1Гб/с.
SPD – это небольшая микросхема, установленная в модуле памяти SD RAM DIMM и содержащая подробную информацию о типе установленной памяти и некоторые другие устройства. PC133 SDRAM(SynchronousDynamicRandomAccessMemory) самая быстрая из класса классической ОП. (были и PC66, PC100). Теперь это самый медленный тип ОЗУ. Физически представляет собой массив микроскопических конденсаторов, «упакованных» в микросхемы памяти. Логически каждый конденсатор есть не что иное, как элементарная однобитовая информационная ячейка с 2-мя состояниями: 0 – если конденсатор не заряжен, 1 – если заряжен. Эти ячейки объединяются в двумерную матрицу, где каждая ячейка адресуется номерами строки и столбца, на пересечении которых она находится. К микросхеме подводятся шины командная (передает команды, управляющие работой микросхем ОП), адресная (адреса строк и столбцов), и данных. Все три синхронизируются импульсами одной и той же частоты. (133). SDRAM – синхронная память и логика работы микросхем памяти этого типа жестко синхронизируется с тактовым сигналом. Например, контроллер памяти точно знает, в течение скольких тактов микросхемы памяти будут готовить запрошенные данные для передачи и на каком такте начнется собственно их передача. Сегодня данная микросхема встречается редко.
Rambus (RD RAM)Двухканальная ОП (микросхема фирмы Intel). DirectRambus – это новая шина памяти, в которой управление адресацией отделено от работы с данными. Система состоит из контроллера DirectRambus, подсоединенного к одному или нескольким модулям DirectRambus DRAM, которые называются RIMM, в отличии от обычных микросхем памяти, соединяемых параллельно, RIMM соединяются последовательно. Канал DirectRambus включает двунаправленную шину данных и шину адреса, т.е. адреса памяти передаются одновременно с данными. Каждая микросхема RDRAM может содержать до 32 независимых банков, SD RAM – от 2 до 8. Свободно работает на высоких тактовых частотах.
Микросхема на 184 контакта Микросхемы ОП с тактовой частотой от 600 до 800 МГц. Когда используется микросхема PC800 (частота синхронизации 400 МГц), пропускная способность шины «память-процессор» достигает 3,2 Гб/с. При использовании PC600 (300 МГц) этот параметр = 2,6 Гб/с.
В свободные гнезда памяти Rambus необходимо устанавливать заглушки ContinuityRimm (CRIMM). Без них система не станет работать, поскольку модули в обоих каналах Rambus включаются каскадно, то есть тактовые и управляющие сигналы проходят через разъемы Rimm последовательно. Емкость ОЗУ может быть до 3 Гб.
Обеспечивают значительное быстродействие при выполнении сложных приложений на ПК и рабочих станциях. Вопрос о быстродействии ОП сегодня очень спорный.
DDR SDRAM (DoubleDataRate) – двойная скорость передачи данных – это по сути модификации обычной SDRAM и отличается от нее тем, что в ней запись и чтение данных происходят и по переднему и по заднему фронту тактового импульса. Поэтому за один такт по шине передается вдвое больше данных, и ее эффективная частота оказывается вдвое больше физической.
2х канальная память DDR266 DDR333 и DDR400 и системы с ней не уступают памяти RDRAM. ОП с удвоенной скоростью передачи данных, а иначе называется PC200 и PC266 в зависимости от тактовой частоты системной шины. Не столь дорогая, чем (3) и явно способствует повышению быстродействия ПК в отличие от (2). В основном благодаря

использованию этой памяти ПК на базе Athlon 1,2 Ггц обошел на многих тестах 1,5 Ггц P-IV с памятью RD RAM.
Сегодня, пока, покупатель не может просто выбрать желательный для него тип ОП, так как она связана с интегральной схемой на системной плате, а та с ЦП. Так, пока, P-IV работает с набором ИС- 850 компании Intel и дорогостоящей памятью RD RAM. (В середине 2001 года планируется появление микросхем, совместимых с устройствами SD RAM и DDR). Если вы хотите приобрести P-IV, то автоматически будете вынуждены приобрести и дорогую ОП. Наборы интегральных схем семейства Athlon используют ОП SD RAM и DDR, но не могут RD RAM.

2) Процессор. Функции, параметры, структура процессора

Основные функции любого процессора следующие:
1)выборка (чтение) выполняемых команд;
2)ввод (чтение) данных из памяти или УВВ;
3)вывод (запись) данных в память или УВВ;
4)обработка данных (операндов), в том числе арифметические операции над ними;
5)адресация памяти, т. е. задание адреса памяти, с которым будет производиться обмен;
6)обработка прерываний и режима прямого доступа к памяти (ПДП).
Важнейшая характеристика процессора-разрядность.Разрядность ШД-скорость работы системы.Разрядность ША-допустимая сложность системы.Кол-во линий управления определяет разнообразие режимов обмена и эффективность обмена процессора с другими устройствами системы.
Магистраль (системная шина) включает в себя три многоразрядные шины:шину данных, шину адреса,шину управления.
Шина данных служит для пересылки данных между ЦП и памятью или ЦП и устройствами ввода/вывода.Шинаадреса .Выбор устройства или ячейки памяти, куда пересылаются или откуда считываются данные по шине данных, производит процессор.Каждое устройство или ячейка оперативной памяти имеет свой адрес. Адрес передается по адресной шине, причем сигналы по ней передаются в одном направлении - от процессора к оперативной памяти и устройствам (однонаправленная шина).
По шине управления передаются управляющие сигналы, определяющие характер обмена

информацией по магистрали и предназначенные памяти и устройствам ввода/вывода.
Микросхема процессора обязательно имеет выводы трех шин: шины адреса, шины данных и шины управления.CLK-подключение внешнего тактового сигнала и тактового резонатора (быстродействие CPU). RESET-сигнал начального сброса. Для подключения CPU к магистрали используют буферные микросхемы, обеспечивающие, если необходимо, демультиплексирвоание сигналов и электрическое буферирование сигналов магистрали. Буферные микросхемы согласуют протоколы шин процессора и магистрали если они не совпадают.
Внутренняя структура микропроцессора. Схема управления выборкой команд, АЛУ, регистры процессора, схема управления прерываниями, схема управления прямым доступом к памяти, логика управления.
Схема управления выборкой команд выполняет чтение команд из памяти и их дешифрацию.
Арифметико-логическое устройство (или АЛУ, ALU) предназначено для обработки информации в соответствии с полученной процессором командой.
Быстродействие АЛУ во многом определяет производительность процессора. Причем важна не только частота тактового сигнала, которым тактируется АЛУ, но и количество тактов, необходимое для выполнения той или иной команды.
Для повышения производительности разработчики стремятся довести время выполнения команды до одного такта, а также обеспечить работу АЛУ на возможно более высокой

частоте. Другой путь повышения производительности процессора — использование нескольких параллельно работающих АЛУ.
Регистры процессора представляют собой по сути ячейки очень быстрой памяти и служат для временного хранения различных кодов: данных, адресов, служебных кодов. Операции с этими кодами выполняются предельно быстро, поэтому, в общем случае, чем больше внутренних регистров, тем лучше. Кроме того, на быстродействие процессора сильно влияет разрядность регистров. Именно
разрядность регистров и АЛУ называется внутренней разрядностью процессора, которая может не совпадать с внешней разрядностью.
По отношению к назначению внутренних регистров существует два основных подхода. Первого придерживается, например, компания Intel, которая каждому регистру отводит строго определенную функцию. С одной стороны, это упрощает организацию процессора и уменьшает время выполнения команды, но с другой — снижает гибкость, а иногда и замедляет работу программы. Второй подход состоит в том, чтобы все (или почти все)регистры сделать равноправными, как , например, в 16-разрядных процессорах Т-11 фирмы DEC. При этом достигается высокая гибкость, но необходимо усложнение структуры процессора.PSW-содержит информацию о выполнении пред-й команды.
Среди общих регистров имеются регистры специального назначения: указатель стека SP (StackPointer), счетчик команд PC (ProgramCounter)

Схема управления прерываниями обрабатывает поступающий на процессор запрос прерывания, определяет адрес начала программы обработки прерывания (адрес вектора прерывания), обеспечивает переход к этой программе после выполнения текущей команды и сохранения в памяти (в стеке) текущего состояния регистров процессора. По окончании программы обработки прерывания процессор возвращается к прерванной программе с восстановленными из памяти (изстека) значениями внутренних регистров.
Схема управления прямым доступом к памяти служит для временного отключения процессора от внешних шин и приостановки работы процессора на время предоставления прямого доступа запросившему его устройству.
Логика управления организует взаимодействие всех узлов процессора, перенаправляет данные, синхронизирует работу процессора с внешними сигналами, а также реализует процедуры ввода и вывода информации.

1. 11 Лекция №26-27 (4 часа).
Тема: «Организация системы прерываний. Организация перехода к прерывающей программе. Принципы организации ввода-вывода**»**

1.11.1 Вопросы лекции:

1) Введение в язык Ассемблера.
2) Язык Ассемблера. Основные команды.

1.11.2 Краткое содержание вопросов:

1) Введение в язык Ассемблера.

Для начала разберёмся с терминологией.
Машинный код – система команд конкретной вычислительной машины (процессора), которая интерпретируется непосредственно процессором. Команда, как правило, представляет собой целое число, которое записывается в регистр процессора. Процессор читает это число и выполняет операцию, которая соответствует этой команде
Язык программирования низкого уровня (низкоуровневый язык программирования) – это язык программирования, максимально приближённый к программированию в машинных кодах. В отличие от машинных кодов, в языке низкого уровня каждой команде соответствует не число, а сокращённое название команды (мнемоника). Например,

команда ADD – это сокращение от слова ADDITION (сложение). Поэтому использование языка низкого уровня существенно упрощает написание и чтение программ (по сравнению с программированием в машинных кодах). Язык низкого уровня привязан к конкретному процессору. Например, если вы написали программу на языке низкого уровня для процессора PIC, то можете быть уверены, что она не будет работать с процессором AVR.
Язык программирования высокого уровня – это язык программирования, максимально приближённый к человеческому языку (обычно к английскому, но есть языки программирования на национальных языках, например, язык 1С основан на русском языке). Язык высокого уровня практически не привязан ни к конкретному процессору, ни к операционной системе (если не используются специфические директивы).
Язык ассемблера – это низкоуровневый язык программирования, на котором вы пишите свои программы. Для каждого процессора существует свой язык ассемблера.
Ассемблер – это специальная программа, которая преобразует (компилирует) исходные тексты вашей программы, написанной на языке ассемблера, в исполняемый файл (файл с расширением EXE или COM). Если быть точным, то для создания исполняемого файла требуются дополнительные программы, а не только ассемблер. Но об этом позже…
В большинстве случаев говорят «ассемблер», а подразумевают «язык ассемблера». Теперь вы знаете, что это разные вещи и так говорить не совсем правильно. Хотя все программисты вас поймут.
ВАЖНО!
В отличие от языков высокого уровня, таких, как Паскаль, Бейсик и т.п., для КАЖДОГО АССЕМБЛЕРА существует СВОЙ ЯЗЫК АССЕМБЛЕРА. Это правило в корне отличает язык ассемблера от языков высокого уровня. Исходные тексты программы (или просто «исходники»), написанной на языке высокого уровня, вы в большинстве случаев можете откомпилировать разными компиляторами для разных процессоров и разных операционных систем. С ассемблерными исходниками это сделать будет намного сложнее. Конечно, эта разница почти не ощутима для разных ассемблеров, которые предназначены для одинаковых процессоров. Но в том то и дело, что для КАЖДОГО ПРОЦЕССОРА существует СВОЙ АССЕМБЛЕР и СВОЙ ЯЗЫК АССЕМБЛЕРА. В этом смысле программировать на языках высокого уровня гораздо проще. Однако за все удовольствия надо платить. В случае с языками высокого уровня мы можем столкнуться с такими вещами как больший размер исполняемого файла, худшее быстродействие и т.п.

В этой книге мы будем говорить только о программировании для компьютеров с процессорами Intel (или совместимыми). Для того чтобы на практике проверить приведённые в книге примеры, вам потребуются следующие программы (или хотя бы некоторые из них):
Emu8086. Хорошая программа, особенно для новичков. Включает в себя редактор исходного кода и некоторые другие полезные вещи. Работает в Windows, хотя программы пишутся под DOS. К сожалению, программа стоит денег (но оно того стоит))).
TASM – Турбо Ассемблер от фирмы Borland. Можно создавать программы как для DOS так и для Windows. Тоже стоит денег и в данный момент уже не поддерживается (да и фирмы Borland уже не существует). А вообще вещь хорошая.
MASM – Ассемблер от компании Microsoft (расшифровывается как МАКРО ассемблер, а не MicrosoftAssembler, как думают многие непосвящённые). Пожалуй, самый популярный ассемблер для процессоров Intel. Поддерживается до сих пор. Условно бесплатная программа. То есть, если вы будете покупать её отдельно, то она будет стоить денег. Но она доступна бесплатно подписчикам MSDN и входит в пакет программ VisualStudio от Microsoft.

WASM – ассемблер от компании Watcom. Как и все другие, обладает преимуществами и недостатками.
Debug - обладает скромными возможностями, но имеет большой плюс - входит в стандартный набор Windows. Поищите ее в папке WINDOWS\COMMAND или WINDOWS\SYSTEM32. Если не найдете, тогда в других папках каталога WINDOWS.
Желательно также иметь какой-нибудь шестнадцатеричный редактор. Не помешает и досовский файловый менеджер, например Волков Коммандер (VC) или Нортон Коммандер (NC). С их помощью можно также посмотреть шестнадцатеричные коды файла, но редактировать нельзя. Бесплатных шестнадцатеричных редакторов в Интернете довольно много. Вот один из них: McAfeeFileInsight v2.1. Этот же редактор можно использовать для работы с исходными текстами программ. Однако мне больше нравится делать это с помощью следующего редактора:
Текстовый редактор. Необходим для написания исходных текстов ваших программ. Могу порекомендовать бесплатный редактор PSPad, который поддерживает множество языков программирования, в том числе и язык Ассемблера.
Все представленные в этой книге программы (и примеры программ) проверены на работоспособность. И именно эти программы используются для реализации примеров программ, приведённых в данной книге.
И еще – исходный код, написанный, например для Emu8086, будет немного отличаться от кода, написанного, например, для TASM. Эти отличия будут оговорены.
Большая часть программ, приведённых в книге, написана для MASM. Во-первых, потому что этот ассемблер наиболее популярен и до сих пор поддерживается. Во-вторых, потому что он поставляется с MSDN и с пакетом программ VisualStudio от Microsoft. Ну и в третьих, потому что я являюсь счастливым обладателем лицензионной копии MASM.
Если же у вас уже есть какой-либо ассемблер, не вошедший в перечисленный выше список, то вам придётся самостоятельно разобраться с его синтаксисом и почитать руководство пользователя, чтобы научиться правильно с ним работать. Но общие рекомендации, приведённые в данной книге, будут справедливы для любых (ну или почти для любых) ассемблеров.

2)Язык Ассемблера. Основные команды.

Язык ассемблера — тип языка программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком. Часто для краткости его называют просто ассемблером, что не верно.
Содержание языка
Команды языка ассемблера один в один соответствуют командам процессора и, фактически, представляют собой удобную символьную форму записи (мнемокод) команд и их аргументов. Также язык ассемблера обеспечивает базовые программные абстракции: связывание частей программы и данных через метки с символьными именами (при ассемблировании для каждой метки высчитывается адрес, после чего каждое вхождение метки заменяется на этот адрес) и директивы.
Директивы ассемблера позволяют включать в программу блоки данных (описанные явно или считанные из файла); повторить определённый фрагмент указанное число раз; компилировать фрагмент по условию; задавать адрес исполнения фрагмента, отличный от адреса расположения в памяти[уточнить!]; менять значения меток в процессе компиляции; использовать макроопределения с параметрами и др.
Каждая модель процессора, в принципе, имеет свой набор команд и соответствующий ему язык (или диалект) ассемблера.
Достоинства и недостатки

Достоинства языка ассемблера

Минимальное количество избыточного кода, то есть использование меньшего количества команд и обращений в память, позволяет увеличить скорость и уменьшить размер программы.

Обеспечение полной совместимости и максимального использования возможностей нужной платформы: использование специальных инструкций и технических особенностей данной платформы.

При программировании на ассемблере становятся доступными специальные возможности: непосредственный доступ к аппаратуре, портам ввода-вывода и особым регистрам процессора, а также возможность написания самомодифицирующегося кода (то есть метапрограммирование, причём без необходимости программного интерпретатора). Последние технологии безопасности, внедряемые в операционные системы, не позволяют делать самомодифицирующегося кода, так как исключают одновременную возможность исполнения инструкций и запись в одном и том же участке памяти (технология W^X в BSD-системах, DEP в Windows).

Недостатки языка ассемблера

Большие объёмы кода и большое число дополнительных мелких задач, что приводит к тому, что код становится очень сложно читать и понимать, а следовательно усложняется отладка и доработка программы, а также трудность реализации парадигм программирования и любых других соглашений. что приводит к сложности совместной разработки.

Меньшее количество доступных библиотек, их малая совместимость между собой.

Непереносимость на другие платформы (кроме двоично совместимых).

Применение

Напрямую вытекает из достоинств и недостатков.

Поскольку большие программы на ассемблере писать крайне неудобно, их пишут на языках высокого уровня. На ассемблере же пишут небольшие фрагменты или модули, для которых критически важны:

быстродействие (драйверы);

размер кода (загрузочные сектора, программное обеспечение для микроконтроллеров и процессоров с ограниченными ресурсами, вирусы, программные защиты);

специальные возможности: работа напрямую с аппаратурой или машинным кодом, то есть загрузчики операционных систем, драйверы, вирусы, системы защиты.

Связывание ассемблерного кода с другими языками

Поскольку на ассемблере чаще всего пишут лишь фрагменты программы, их необходимо связывать с остальными частями на других языках. Это достигается 2 основными способами:

На этапе компиляции — вставка в программу ассемблерных фрагментов (англ. inlineassembler) специальными директивами языка, в том числе написание процедур на языке ассемблера. Способ удобен для несложных преобразований данных, но полноценного ассемблерного кода, с данными и подпрограммами, включая подпрограммы с множеством входов и выходов, не поддерживаемых высокоуровневыми языками, с помощью него сделать нельзя.

На этапе компоновки, или раздельная компиляция. Для взаимодействия скомпонованных модулей достаточно, чтобы связующие функции[3] поддерживали нужные соглашения о вызовах (англ. callingconventions) и типы данных. Написаны же отдельные модули могут быть на любых языках, в том числе и на ассемблере.

Синтаксис

Общепринятого стандарта для синтаксиса языков ассемблера не существует. Однако, существуют стандарты, которых придерживаются большинство разработчиков языков ассемблера. Основными такими стандартами являются Intel-синтаксис и AT&T-синтаксис.

Инструкции

Общий формат записи инструкций одинаков для обоих стандартов:
[метка:] опкод [операнды] [;комментарий]
где опкод — непосредственно мнемоника инструкции процессору. К ней могут быть добавлены префиксы (повторения, изменения типа адресации и пр.).
В качестве операндов могут выступать константы, названия регистров, адреса в оперативной памяти и пр.. Различия между стандартами Intel и AT&T касаются, в основном, порядка перечисления операндов и их синтаксиса при различных методах адресации.
Используемые мнемоники обычно одинаковы для всех процессоров одной архитектуры или семейства архитектур (среди широко известных - мнемоники процессоров и контроллеров Motorola, ARM, x86). Они описываются в спецификации процессоров.
Возможные исключения:
Если ассемблер использует кроссплатформенный AT&T-синтаксис (оригинальные мнемоники приводятся к синтаксису AT&T)
Если изначально существовало два стандарта записи мнемоник (система команд была наследована от процессора другого производителя).
Например, процессор Zilog Z80 наследовал систему команд Intel i8080, расширил ее и поменял мнемоники (и обозначения регистров) на свой лад. Например сменил интеловские mov[4] на ld. Процессоры MotorolaFireball наследовали систему команд Z80, несколько её урезав. Вместе с тем, Motorola официально вернулась к мнемоникам Intel. И в данный момент половина ассемблеров для Fireball работает с интеловскими мнемониками, а половина с мнемониками Zilog.
Директивы
Кроме инструкций, программа может содержать директивы: команды, не переводящиеся непосредственно в машинные инструкции, а управляющие работой компилятора. Набор и синтаксис их значительно разнятся и зависят не от аппаратной платформы, а от используемого компилятора (порождая диалекты языков в пределах одного семейства архитектур). В качестве "джентельменского набора" директив можно выделить:
определение данных (констант и переменных)
управление организацией программы в памяти и параметрами выходного файла
задание режима работы компилятора
всевозможные абстракции (т.е. элементы языков высокого уровня) - от оформления процедур и функций (для упрощения реализации парадигмы процедурного программирования) до условных конструкций и циклов (для парадигмы структурного программирования)
макросы
Пример программы
Пример программы Helloworld для MS-DOS для архитертуры x86 на диалекте TASM:

```
.MODEL TINY
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
ORG 100h
START:
mov ah,9
movdx,OFFSETMsg
int 21h
int 20h
Msg DB 'Hello World',13,10,'$'
CODE ENDS
END START
```

1. 12 Лекция №28-30 (6 часов).

Тема: «Архитектура системной платы. Установка и конфигурирование компонентов»

1.12.1 Вопросы лекции:

1) Загрузочный сектор.
2) Файловая система FAT.

1.12.2 Краткое содержание вопросов:

1) Загрузочный сектор.

Загрузочный сектор, бутсектор (англ. bootsector, Volumebootsector (Volumebootrecord), Partitionbootsector) — это особый сектор на жёстком диске,дискете или другом дисковом устройстве хранения информации. (Для дискеты это первый физический сектор, для жёсткого диска — первый физический сектор для каждого раздела.) В процессе загрузки компьютера с дискеты он загружается в память программой POST (в компьютерах архитектуры IBM PC обычно с адреса 0000:7c00), ему передается управление командой longjump.

Загрузочный сектор, иногда называемый stage1 (то есть первым этапом загрузки операционной системы), загружает программу второго этапа загрузки операционной системы stage2 (вторичный загрузчик, иногда в качестве stage2 загружается bootmanager или программа авторизации и защиты доступа). В некоторых ОС роль stage1 выполняет MBR, и при загрузке ОС с жёсткого диска загрузочный сектор не используется.

2)Файловая система FAT.

Файловая система FAT (FileAllocationTable) была разработана Биллом Гейтсом и Марком Макдональдом в 1977 году.

Сейчас существуют три типа файловой системы FAT:

FAT12 – поддерживает очень небольшие объемы дисков, поэтому сейчас она применяется только на дискетах.

FAT16 – используется на винчестерах и поддерживает диски объемом до 2 Гб, поэтому сейчас данная файловая система практически не используется.

FAT32 – теоретически поддерживаются диски объемом до 2 Тб. Поддерживается, начиная с операционной системы Windows 95 OSR2. Данная файловая система сейчас достаточно популярна, хотя в последние годы многие пользователи Windows XP предпочитают использовать NTFS (NewTechnologyFileSystem).

Структура

Загрузочный сектор	Таблица FAT (1-я копия)	Таблица FAT (2-я копия)	Корневой каталог	Область данных

Рис. 1. Структура файловой системы FAT

– Загрузочный сектор

В начале раздела диска с файловой системой FAT располагается загрузочный сектор. Он необходим для начальной загрузки компьютера. Так же в нем располагается информация о параметрах данного раздела.

– Таблицаразмещенияфайлов (File Allocation Table)

Вся область данных диска разделена на кластеры – блоки, размер которых задается при форматировании диска. На дискете, например, размер кластера равен 512-ти байтам. А на современных винчестерах с объемом диска более 32 Гб размер кластера равен 32 Кб. Каждый файл и каталог занимает один или несколько кластеров. Таким образом, образуются цепочки кластеров.

В таблице размещения файлов каждый кластер помечается специальным образом. Размер метки в битах для каждого кластера указывается в названии файловой системы. Т.е. для файловой системы FAT16 размер метки будет равен 16-ти байтам, для FAT32 – 32-м и т.д. Всего существует три типа меток для кластеров:

Свободный кластер – кластер, в который будут записываться новые файлы и каталоги.
Занятый кластер – в метке указывается следующий кластер в цепочке. Если цепочка кластеров заканчивается, то кластер помечается особой меткой.
BAD-блок – кластер с ошибками доступа. Помечается при форматировании диска, что бы исключить в последующем доступ к нему.
Повреждение таблицы размещения файлов полностью уничтожает структуру файловой системы, поэтому на диске всегда хранится две копии таблицы.
– Корневой каталог
Область диска, в котором располагается информация о корневом каталоге. Размер ее ограничен, поэтому в корневом каталоге диска может находиться не более 512-ти файлов и подкаталогов.
– Область данных
Оставшаяся часть раздела, на которой размещается содержимое файлов и каталогов.
FAT32 и NTFS
Сейчас основным конкурентом популярной файловой системы FAT32 является NTFS. Файловая система NTFS получила популярность с распространением операционной системы Windows XP.
– FAT32
Основным преимуществом этой файловой системы является ее простота и совместимость со старыми операционными системами. Для этой файловой системы существует большое количество подробной документации. Существует ограничение на максимальный размер файла – 4 Гб. Сбои в системе часто приводят к повреждением одного или нескольких файлов. Однако, при серьезных повреждениях, восстановить информацию гораздо проще, чем в случае с NTFS.
– NTFS
Основными преимуществами файловой системы NTFS являются ее защищенность от несанкционированного доступа. В этой файловой системе отсутствуют ограничения на размер файлов и каталогов. Так же ее особенность является журналирование – запись всех операций перед их выполнением в специальный журнал. В случае, если во время выполнения операций с файловой системой произойдет сбой (зависание операционной системы, отключение электричества и т.п.), то она на основе записей в журнале сможет вернуть себя в прежнее состояние. Однако, в случае серьезного сбоя восстановить информацию будет очень сложно, подчас невозможно. Причиной тому является отсутствие официальной документации файловой системы от Microsoft. Так же недостатком NTFS является несовместимость со старыми версиями операционных систем (Windows 95, 98).

1. 13 Лекция №31-33 (6 часов).
Тема: «Шины расширения. Шина USB**»**

1.13.1 Вопросы лекции:

1) Интерфейсы COM
2) Интерфейсы USB

1.13.2 Краткое содержание вопросов:

1) Флэш-носители.

Флеш-память (англ. *flashmemory*) — разновидность полупроводниковой технологии электрически перепрограммируемой памяти (EEPROM). Это же слово используется в электронной схемотехнике для обозначения технологически законченных решений постоянных запоминающих устройств в виде микросхем на базе этой

полупроводниковой технологии. В быту это словосочетание закрепилось за широким классом твердотельных устройств хранения информации.

Благодаря компактности, дешевизне, механической прочности, большому объёму, скорости работы и низкому энергопотреблению, флеш-память широко используется в цифровых портативных устройствах и носителях информации. Серьёзным недостатком данной технологии является ограниченный срок эксплуатации носителей,[1][2] а также чувствительность к электростатическому разряду.

2) Дисковая память.

Носителями информации являются поверхности гибких и жестких дисков, в качестве немагнитных основ которых используются соответственно майлар (как и в магнитных лентах) и алюминиевые (в ряде случаев стеклянные) круги (диски). Стеклянные диски являются менее критичными к температурным изменениям и позволяют увеличить плотность записи информации. В настоящее время наиболее широкое распространение получили диски с напыленным магнитным слоем, а точнее, с металлической пленкой (например, кобальт).

Перед осуществлением записи на магнитный диск он должен быть специальным образом инициализирован – отформатирован. В результате форматирования на поверхности образуются концентрические окружности (синхронизирующие метки диска), называемые **дорожками** (track). Количество дорожек зависит от типа диска. Дорожки разбиваются на участки фиксированной длины, называемые **секторами.** Количество секторов на дорожке определяется типом и **форматом** диска, и они в основном одинаковы для всех дорожек. IBM PC-совместимые ПК могут работать с несколькими размерами секторов от 128 до 1024 байт. Стандартным сектором считается сектор из 512 байт. Данные любого размера (разрядности) размещаются в секторах с фиксированным размером, а дисковые операции записи и считывания производятся с целыми секторами.

Дорожки и сектора нумеруются с нуля, начиная с внешнего края диска, при этом сектор с нулевым номером на каждой дорожке резервируется для системных целей. Диски имеют две стороны. Так как накопители на жестких дисках могут состоять из нескольких дисков (стопка), то совокупность всех дорожек, по одной на каждой стороне с одинаковыми номерами, образует **цилиндр** с номером соответствующей дорожки.

1. 14 Лекция №34-36 (6 часов).
Тема: «Параллельный интерфейс. Последовательный интерфейс**»**

1.14.1 Вопросы лекции:

1) История возникновения и развития вычислительной техники
2) Классификация компьютеров. Поколения вычислительной техники

1.14.2 Краткое содержание вопросов:

1) История возникновения и развития вычислительной техники

Первым устройством, предназначенным для облегчения счета, были счеты. С помощью костяшек счетов можно было совершать операции сложения и вычитания и несложные умножения.

1642 г. — французский математик Блез Паскаль сконструировал первую механическую счетную машину «Паскалина», которая могла механически выполнять сложение чисел.

1673 г. — Готфрид Вильгельм Лейбниц сконструировал арифмометр, позволяющий механически выполнять четыре арифметических действия.

Первая половина XIX в. — английский математик Чарльз Бэббидж попытался построить универсальное вычислительное устройство, то есть компьютер. Бэббидж называл его аналитической машиной. Он определил, что компьютер должен содержать память и управляться с помощью программы. Компьютер по Бэббиджу — это механическое устройство, программы для которого задаются посредством перфокарт — карт из плотной бумаги с информацией, наносимой с помощью отверстий (они в то время уже широко употреблялись в ткацких станках).

1941 г. — немецкий инженер Конрад Цузе построил небольшой компьютер на основе нескольких электромеханических реле.

1943 г. — в США на одном из предприятий фирмы IBM ГовардЭйкен создал компьютер под названием «Марк-1». Он позволял проводить вычисления в сотни раз быстрее, чем вручную (с помощью арифмометра), и использовался для военных расчетов. В нем использовалось сочетание электрических сигналов и механических приводов. «Марк-1» имел размеры: 15 * 2—5 м и содержал 750 000 деталей. Машина была способна перемножить два 32-разрядных числа за 4 с.

1943 г. — в США группа специалистов под руководством Джона Мочли и ПроспераЭкерта начала конструировать компьютер ENIAC на основе электронных ламп.

1945 г. — к работе над ENIAC был привлечен математик Джон фон Нейман, который подготовил доклад об этом компьютере. В своем докладе фон Нейман сформулировал общие принципы функционирования компьютеров, т. е. универсальных вычислительных устройств. До сих пор подавляющее большинство компьютеров сделано в соответствии с теми принципами, которые изложил Джон фон Нейман.

1947 г. — Экертом и Мочли начата разработка первой электронной серийной машины UNIVAC (UniversalAutomaticComputer). Первый образец машины (UNIVAC-1) был построен для бюро переписи США и пущен в эксплуатацию весной 1951 г. Синхронная, последовательного действия вычислительная машина UNIVAC-1 была создана на базе ЭВМ ENIAC и EDVAC. Работала она с тактовой частотой 2,25 МГц и содержала около 5000 электронных ламп. Внутреннее запоминающее устройство емкостью 1000 12-разрядных десятичных чисел было выполнено на 100 ртутных линиях задержки.

1949 г. — английским исследователем Морнсом Уилксом построен первый компьютер, в котором были воплощены принципы фон Неймана.

1951 г. — Дж. Форрестер опубликовал статью о применении магнитных сердечников для хранения цифровой информации, В машине «Whirlwind-1» впервые была применена память на магнитных сердечниках. Она представляла собой 2 куба с 32-32-17

сердечниками, которые обеспечивали хранение 2048 слов для 16-разрядных двоичных чисел с одним разрядом контроля на четность.

1952 г. — фирма IBM выпустила свой первый промышленный электронный компьютер IBM 701, который представлял собой синхронную ЭВМ параллельного действия, содержащую 4000 электронных ламп и 12 000 диодов. Усовершенствованный вариант машины IBM 704 отличался высокой скоростью работы, в нем использовались индексные регистры и данные представлялись в форме с плавающей запятой.

После ЭВМ IBM 704 была выпущена машина IBM 709, которая в архитектурном плане приближалась к машинам второго и третьего поколений. В этой машине впервые была применена косвенная адресация и впервые появились каналы ввода — вывода.

1952 г. — фирма RemingtonRand выпустила ЭВМ UNIVAC-t 103, в которой впервые были применены программные прерывания. Сотрудники фирмы RemingtonRand использовали алгебраическую форму записи алгоритмов под названием «ShortCode» (первый интерпретатор, созданный в 1949 г. Джоном Мочли).

1956 г. — фирмой IBM были разработаны плавающие магнитные головки на воздушной подушке. Изобретение их позволило создать новый тип памяти — дисковые запоминающие устройства (ЗУ), значимость которых была в полной мере оценена в последующие десятилетия развития вычислительной техники. Первые ЗУ на дисках появились в машинах IBM 305 и RAMAC. Последняя имела пакет, состоявший из 50 металлических дисков с магнитным покрытием, которые вращались со скоростью 12000 об. /мин. На поверхности диска размещалось 100 дорожек для записи данных, по 10 000 знаков каждая.

1956 г. — фирма Ferranti выпустила ЭВМ «Pegasus», в которой впервые нашла воплощение концепция регистров общего назначения (РОН). С появлением РОН устранено различие между индексными регистрами и аккумуляторами, и в распоряжении программиста оказался не один, а несколько регистров-аккумуляторов.

1957 г. — группа под руководством Д. Бэкуса завершила работу над первым языком программирования высокого уровня, получившим название ФОРТРАН. Язык, реализованный впервые на ЭВМ IBM 704, способствовал расширению сферы применения компьютеров.

1960-е гг. — 2-е поколение ЭВМ, логические элементы ЭВМ реализовываются на базе полупроводниковых приборов-транзисторов, развиваются алгоритмические языки программирования, такие как Алгол, Паскаль и другие.

1970-е гг. — 3-е поколение ЭВМ, интегральные микросхемы, содержащие на одной полупроводниковой пластине тысячи транзисторов. Начали создаваться ОС, языки структурного программирования.

1974 г. — несколько фирм объявили о создании на основе микропроцессора Intel-8008 персонального компьютера — устройства, выполняющего те же функции, что и большой компьютер, но рассчитанного на одного пользователя.

1975 г. — появился первый коммерчески распространяемый персональный компьютер Альтаир-8800 на основе микропроцессора Intel-8080. Этот компьютер имел оперативную память всего 256 байт, клавиатура и экран отсутствовали.

Конец 1975 г. — Пол Аллен и Билл Гейтс (будущие основатели фирмы Microsoft) создали для компьютера «Альтаир» интерпретатор языка Basic, позволивший пользователям просто общаться с компьютером и легко писать для него программы.

Август 1981 г. — компания IBM представила персональный компьютер IBM PC. В качестве основного микропроцессора компьютера использовался 16-разрядный микропроцессор Intel-8088, который позволял работать с 1 мегабайтом памяти.

1980-е гг. — 4-е поколение ЭВМ, построенное на больших интегральных схемах. Микропроцессоры реализовываются в виде единой микросхемы, Массовое производство персональных компьютеров.

1990-е гг. — 5-е поколение ЭВМ, сверхбольшие интегральные схемы. Процессоры содержат миллионы транзисторов. Появление глобальных компьютерных сетей массового пользования.

2000-е гг. — 6-е поколение ЭВМ. Интеграция ЭВМ и бытовой техники, встраиваемые компьютеры, развитие сетевых вычислений.

2) Классификация компьютеров. Поколения вычислительной техники

Компьютеры могут быть классифицированы по разным признакам, например по габаритам, по областям применения, по быстродействию, по функциям, по этапам создания и еще по многим другим параметрам.

Мы рассмотрим классификацию по обобщенному параметру, где в разной степени учтено несколько характерных признаков:

назначение и роль компьютеров в системе обработки информации;
условия взаимодействия человека и компьютера;
габариты компьютера;
ресурсные возможности компьютера.

В соответствии с вышесформулированными признаками и тенденциями развития компьютерной техники можно рассмотреть следующую классификацию компьютеров:

портативные компьютеры;
микрокомпьютеры, в том числе — персональные компьютеры;
мэйнфреймы (универсальные компьютеры);
суперкомпьютеры.

Портативные компьютеры обычно нужны руководителям предприятий, менеджерам, учёным, журналистам, которым приходится работать вне офиса — дома, на презентациях или во время командировок.

Основные разновидности портативных компьютеров:

Микрокомпьютеры — это компьютеры, в которых центральный процессор выполнен в виде микропроцессора.

Продвинутые модели микрокомпьютеров имеют несколько микропроцессоров. Производительность компьютера определяется не только характеристиками применяемого микропроцессора, но и ёмкостью оперативной памяти, типами периферийных устройств, качеством конструктивных решений и др.

Микрокомпьютеры представляют собой инструменты для решения разнообразных сложных задач. Их микропроцессоры с каждым годом увеличивают мощность, а

периферийные устройства — эффективность. Быстродействие — порядка 1 - 10 миллионов операций в сек.

Разновидность микрокомпьютера — микроконтроллер. Это основанное на микропроцессоре специализированное устройство, встраиваемое в систему управления или технологическую линию.

В класс персональных компьютеров входят различные машины — от дешёвых домашних и игровых с небольшой оперативной памятью, с памятью программы на кассетной ленте и обычным телевизором в качестве дисплея, до сверхсложных машин с мощным процессором, винчестерским накопителем ёмкостью в десятки Гигабайт, с цветными графическими устройствами высокого разрешения, средствами мультимедиа и другими дополнительными устройствами.

Персональныйкомпьютер должен удовлетворять следующим требованиям:

стоимость от нескольких сотен до 5-10 тысяч долларов;

наличие внешних ЗУ на магнитных дисках;

объём оперативной памяти не менее 4 Мбайт;

наличие операционной системы;

способность работать с программами на языках высокого уровня;

ориентация на пользователя-непрофессионала (в простых моделях).

Мейнфрейм - это синоним понятия "большая универсальная ЭВМ". Мейнфреймы и до сегодняшнего дня остаются наиболее мощными (не считая суперкомпьютеров)вычислительными системами общего назначения, обеспечивающими непрерывный круглосуточный режим эксплуатации. Они могут включать один или несколько процессоров, каждый из которых, в свою очередь, может оснащаться векторными сопроцессорами (ускорителями операций с суперкомпьютерной производительностью). В нашем сознании мейнфреймы все еще ассоциируются с большими по габаритам машинами, требующими специально оборудованных помещений с системами водяного охлаждения и кондиционирования. Однако это не совсем так. Прогресс в области элементно-конструкторской базы позволил существенно сократить габариты основных устройств. Наряду со сверхмощными мейнфреймами, требующими организации двухконтурной водяной системы охлаждения, имеются менее мощные модели, для охлаждения которых достаточно принудительной воздушной вентиляции, и модели, построенные по блочно-модульному принципу и не требующие специальных помещений и кондиционеров.

Основными поставщиками мейнфреймов являются известные компьютерные компании IBM, Amdahl, ICL, SiemensNixdorf и некоторые другие, но ведущая роль принадлежит безусловно компании IBM. Именно архитектура системы IBM/360, выпущенной в 1964 году, и ее последующие поколения стали образцом для подражания. В нашей стране в течение многих лет выпускались машины ряда ЕС ЭВМ, являвшиеся отечественным аналогом этой системы. В архитектурном плане мейнфреймы представляют собоймногопроцессорные системы, содержащие один или несколько центральных и периферийных процессоров с общей памятью, связанных между собой высокоскоростными магистралями передачи данных. При этом основная вычислительная нагрузка ложится на центральные процессоры, а периферийные процессоры (в терминологии IBM - селекторные, блок-мультиплексные, мультиплексные каналы и процессоры телеобработки) обеспечивают работу с широкой номенклатурой периферийных устройств.

Они предназначены для решения широкого класса научно-технических задач и являются сложными и дорогими машинами. Их целесообразно применять в больших системах при наличии не менее 200 - 300 рабочих мест.

Централизованная обработка данных на мэйнфрейме обходится примерно в 5 - 6 раз дешевле, чем распределённая обработка при клиент-серверном подходе.

Известный мейнфрейм S/390 фирмы IBM обычно оснащается не менее чем тремя процессорами. Максимальный объём оперативного хранения достигает 342 Терабайт. Производительность его процессоров, пропускная способность каналов, объём оперативного хранения позволяют наращивать число рабочих мест в диапазоне от 20 до 200000 с помощью простого добавления процессорных плат, модулей оперативной памяти и дисковых накопителей.
Десятки мейнфреймов могут работать совместно под управлением одной операционной системы над выполнением единой задачи.
Отличительной особенностью суперкомпьютеров являются векторные процессоры, оснащенные аппаратурой для параллельного выполнения операций с многомерными цифровыми объектами — векторами и матрицами. В них встроены векторные регистры и параллельный конвейерный механизм обработки. Если на обычном процессоре программист выполняет операции над каждым компонентом вектора по очереди, то на векторном — выдаёт сразу векторые команды.
Векторная аппаратура очень дорога, в частности, потому, что требуется много сверхбыстродействующей памяти под векторные регистры.
Наряду с векторно-конвейерной системой обработки данных существует и скалярная система, основанная на выполнении обычных арифметических операций над отдельными числами или парами чисел.
Строго говоря, системы, использующие скалярную обработку данных, по своей производительности уступают суперЭВМ, но у них наблюдаются тенденции, характерные для высокопроизводительных вычислительных систем: необходимость распараллеливания больших задач между процессорами.
Наиболее распространённые суперкомпьютеры — массово-параллельные компьютерные системы. Они имеют десятки тысяч процессоров, взаимодействующих через сложную, иерархически организованую систему памяти.

Супер-компьютеры используются для решения сложных и больших научных задач (метеорология, гидродинамика и т. п.), в управлении, разведке, в качестве централизованных хранилищ информации и т.д.
Элементная база — микросхемы сверхвысокой степени интеграции.
К суперкомпьютерам часто относят и серверы.

Сервер-мощный компьютер в вычислительных сетях, который обеспечивает обслуживание подключенных к нему компьютеров и выход в другие сети.
В зависимости от назначения определяют такие типы серверов:
Сервер приложений обрабатывает запросы от всех станций вычислительной сети и предоставляет им доступ к общим системным ресурсам (базам данных, библиотекам программ, принткрам, факсам и др.).
Файл-сервер-для работы с базами данных и использования файлов информации, хранящихся в ней.
Архивационный сервер-для резервного копирования информации в крупных многосервисных сетях. Он использует накопители на магнитной ленте(стриммеры) со сменными картриджами емкостью до 5 Гбайт. Обычно выполняет ежедневное автоматическое архивирование информации от подключенных серверов и рабочих станций.
Факс-сервер-для организации эффективной многоадресной факсимильной связи, с несколькими факсмодемными платами, со специальной защитой информации от несанкционированного доступа в процессе передачи, с системой хранения электронных факсов.
Почтовый сервер-то же, что и факс-сервер, но для организации электронной почты, с электронными почтовыми ящиками.
Сервер печати-для эффективного использования системных принтеров.
Сервер- телеконференций-компьютер, имеющий программу обслуживания пользователей

телеконференциями и новостями, он также может иметь систему автоматической обработки видеоизображений и др.
Любой компьютер, если установить на нем соответствующее сетевое программное обеспечение, способен стать сервером. Кроме того, один компьютер одновременно может выполнять несколько функций-быть, к примеру, почтовым сервером, сервером новостей, сервером приложений и т.д.
Существуют различные классификации компьютерной техники:
по этапам развития (по поколениям);
по архитектуре;
по производительности;
по условиям эксплуатации;
по количеству процессоров;
по потребительским свойствам и т.д.
Четких границ между классами компьютеров не существует. По мере совершенствования структур и технологии производства, появляются новые классы компьютеров, границы существующих классов существенно изменяются.

2. . МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ
ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

2.1 Практическое занятие №1-2 (4 часа).
Тема: «Арифметические основы построения и логические основы построения ЭВМ**»**

2.1.1 Задание для работы:

1) История возникновения и развития вычислительной техники
2) Классификация компьютеров. Поколения вычислительной техники.

2.1.2 Краткое описание проводимого занятия:

1) Обработка прерываний. Многозадачность.

Многозадачность, multitasking — свойство операционной системы или среды программирования, обеспечивать возможность параллельной (или псевдопараллельной) обработки нескольких процессов. Истинная многозадачность операционной системы возможна только в распределенных вычислительных системах.

Примитивные многозадачные среды обеспечивают чистое “разделение ресурсов”, когда за каждой задачей закрепляется определённый участок памяти, и задача активизируется в строго определённые интервалы времени.

Более развитые многозадачные системы проводят распределение ресурсов динамически, когда задача стартует в памяти или покидает память в зависимости от её приоритета и от стратегии системы. Такая многозадачная среда обладает следующими особенностями:

- Каждая задача имеет свой приоритет, в соответствии с которым получает время и память
- Система организует очереди задач так, чтобы все задачи получили ресурсы, в зависимости от приоритетов и стратегии системы
- Система организует обработку прерываний, по которым задачи могут активироваться, деактивироваться и удаляться
- По окончании положенного кванта времени задача может временно выбрасываться из памяти, отдавая ресурсы другим задачам, а потом через определённое системой время, восстанавливаться в памяти (свопинг)
- Система обеспечивает защиту памяти от несанкционированного вмешательства других задач

- Система распознаёт сбои и зависания отдельных задач и прекращает их
- Система решает конфликты доступа к ресурсам и устройствам, не допуская тупиковых ситуаций общего зависания от ожидания заблокированных ресурсов
- Система гарантирует каждой задаче, что рано или поздно она будет активирована
- Система обрабатывает запросы реального времени
- Система обеспечивает коммуникацию между процессами

Типы псевдопараллельной многозадачности

1. Невытесняющая многозадачность

   Тип многозадачности, при котором операционная система одновременно загружает в память два или более приложений, но процессорное время предоставляется только основному приложению. Для выполнения фонового приложения оно должно быть активизировано.
2. Совместная или кооперативная многозадачность

   Тип многозадачности, при котором фоновые задачи выполняются только во время простоя основного процесса и только в том случае, если на это получено разрешение основного процесса.

   Кооперативную многозадачность можно назвать многозадачностью “второй ступени” поскольку она использует более передовые методы, чем простое переключение задач, реализованное многими известными программами (например, MS-DOS shell из MS-DOS 5.0 при простом переключении активная программа получает все процессорное время, а фоновые приложения полностью замораживаются. При кооперативной многозадачности приложение может захватить фактически столько процессорного времени, сколько оно считает нужным. Все приложения делят процессорное время, периодически передавая управление следующей задаче.
3. Вытесняющая или приоритетная многозадачность (режим реального времени)

Вид многозадачности, в котором операционная система сама передает управление от одной выполняемой программы другой. Распределение процессорного времени осуществляется планировщиком процессов. Этот вид многозадачности обеспечивает более быстрый отклик на действия пользователя.

Вытесняющая многозадачность — это вид многозадачности при котором планирование процессов основывается на абсолютных приоритетах. Процесс с меньшим приоритетом (например пользовательская программа) может быть вытеснен при его выполнении более приоритетным процессом (например системной или диагностической программой). Иногда этот вид многозадачности называют приоритетным.

Каждая работающая программа имеет свое защищенное адресное пространство. Многопоточное (англ. multithread) выполнение отдельных задач позволяет при задержке в выполнении одного потока не останавливать задачу полностью, а работать со следующим потоком.

2) Технология ввода-вывода. Ввод-вывод с использованием прерываний.

Управление вводом-выводом

Одной из главных функций ОС является управление всеми устройствами ввода-вывода компьютера. ОС должна передавать устройствам команды, перехватывать прерывания и обрабатывать ошибки; она также должна обеспечивать интерфейс между устройствами и остальной частью системы. В целях развития интерфейс должен быть одинаковым для всех типов устройств (независимость от устройств).

Физическая организация устройств ввода-вывода

Устройства ввода-вывода делятся на два типа: *блок-ориентированные* устройства и *байт-ориентированные* устройства. Блок-ориентированные устройства хранят информацию в блоках фиксированного размера, каждый из которых имеет свой собственный адрес. Самое распространенное блок-ориентированное устройство - диск.

Байт-ориентированные устройства не адресуемы и не позволяют производить операцию поиска, они генерируют или потребляют последовательность байтов. Примерами являются терминалы, строчные принтеры, сетевые адаптеры. Однако некоторые внешние устройства не относятся ни к одному классу, например, часы, которые, с одной стороны, не адресуемы, а с другой стороны, не порождают потока байтов. Это устройство только выдает сигнал прерывания в некоторые моменты времени.

Внешнее устройство обычно состоит из механического и электронного компонента. Электронный компонент называется контроллером устройства или адаптером. Механический компонент представляет собственно устройство. Некоторые контроллеры могут управлять несколькими устройствами. Если интерфейс между контроллером и устройством стандартизован, то независимые производители могут выпускать совместимые как контроллеры, так и устройства.

Операционная система обычно имеет дело не с устройством, а с контроллером. Контроллер, как правило, выполняет простые функции, например, преобразует поток бит в блоки, состоящие из байт, и осуществляют контроль и исправление ошибок. Каждый контроллер имеет несколько регистров, которые используются для взаимодействия с центральным процессором. В некоторых компьютерах эти регистры являются частью физического адресного пространства. В таких компьютерах нет специальных операций ввода-вывода. В других компьютерах адреса регистров ввода-вывода, называемых часто портами, образуют собственное адресное пространство за счет введения специальных операций ввода-вывода (например, команд IN и OUT в процессорах i86).

ОС выполняет ввод-вывод, записывая команды в регистры контроллера. Например, контроллер гибкого диска IBM PC принимает 15 команд, таких как READ, WRITE, SEEK, FORMAT и т.д. Когда команда принята, процессор оставляет контроллер и занимается другой работой. При завершении команды контроллер организует прерывание для того, чтобы передать управление процессором операционной системе, которая должна проверить результаты операции. Процессор получает результаты и статус устройства, читая информацию из регистров контроллера.

Организация программного обеспечения ввода-вывода

Основная идея организации программного обеспечения ввода-вывода состоит в разбиении его на несколько уровней, причем нижние уровни обеспечивают экранирование особенностей аппаратуры от верхних, а те, в свою очередь, обеспечивают удобный интерфейс для пользователей.

Ключевым принципом является независимость от устройств. Вид программы не должен зависеть от того, читает ли она данные с гибкого диска или с жесткого диска.

Очень близкой к идее независимости от устройств является идея единообразного именования, то есть для именования устройств должны быть приняты единые правила.

Другим важным вопросом для программного обеспечения ввода-вывода является обработка ошибок. Вообще говоря, ошибки следует обрабатывать как можно ближе к аппаратуре. Если контроллер обнаруживает ошибку чтения, то он должен попытаться ее скорректировать. Если же это ему не удается, то исправлением ошибок должен заняться драйвер устройства. Многие ошибки могут исчезать при повторных попытках выполнения операций ввода-вывода, например, ошибки, вызванные наличием пылинок на головках чтения или на диске. И только если нижний уровень не может справиться с ошибкой, он сообщает об ошибке верхнему уровню.

Еще один ключевой вопрос - это использование блокирующих (синхронных) и неблокирующих (асинхронных) передач. Большинство операций физического ввода-вывода выполняется асинхронно - процессор начинает передачу и переходит на другую работу, пока не наступает прерывание. Пользовательские программы намного легче писать, если операции ввода-вывода блокирующие - после команды READ программа автоматически приостанавливается до тех пор, пока данные не попадут в буфер

программы. ОС выполняет операции ввода-вывода асинхронно, но представляет их для пользовательских программ в синхронной форме.

Последняя проблема состоит в том, что одни устройства являются разделяемыми, а другие - выделенными. Диски - это разделяемые устройства, так как одновременный доступ нескольких пользователей к диску не представляет собой проблему. Принтеры - это выделенные устройства, потому что нельзя смешивать строчки, печатаемые различными пользователями. Наличие выделенных устройств создает для операционной системы некоторые проблемы.

Для решения поставленных проблем целесообразно разделить программное обеспечение ввода-вывода на четыре слоя (рисунок 2.30):

- Обработка прерываний,
- Драйверы устройств,
- Независимый от устройств слой операционной системы,
- Пользовательский слой программного обеспечения.

Рис. 2.30. Многоуровневая организация подсистемы ввода-вывода

Обработка прерываний

Прерывания должны быть скрыты как можно глубже в недрах операционной системы, чтобы как можно меньшая часть ОС имела с ними дело. Наилучший способ состоит в разрешении процессу, инициировавшему операцию ввода-вывода, блокировать себя до завершения операции и наступления прерывания. Процесс может блокировать себя, используя, например, вызов DOWN для семафора, или вызов WAIT для переменной условия, или вызов RECEIVE для ожидания сообщения. При наступлении прерывания процедура обработки прерывания выполняет разблокирование процесса, инициировавшего операцию ввода-вывода, используя вызовы UP, SIGNAL или посылая процессу сообщение. В любом случае эффект от прерывания будет состоять в том, что ранее заблокированный процесс теперь продолжит свое выполнение.

Драйверы устройств

Весь зависимый от устройства код помещается в драйвер устройства. Каждый драйвер управляет устройствами одного типа или, может быть, одного класса.

В операционной системе только драйвер устройства знает о конкретных особенностях какого-либо устройства. Например, только драйвер диска имеет дело с дорожками, секторами, цилиндрами, временем установления головки и другими факторами, обеспечивающими правильную работу диска.

Драйвер устройства принимает запрос от устройств программного слоя и решает, как его выполнить. Типичным запросом является чтение n блоков данных. Если драйвер был свободен во время поступления запроса, то он начинает выполнять запрос немедленно. Если же он был занят обслуживанием другого запроса, то вновь поступивший запрос присоединяется к очереди уже имеющихся запросов, и он будет выполнен, когда наступит его очередь.

Первый шаг в реализации запроса ввода-вывода, например, для диска, состоит в преобразовании его из абстрактной формы в конкретную. Для дискового драйвера это означает преобразование номеров блоков в номера цилиндров, головок, секторов, проверку, работает ли мотор, находится ли головка над нужным цилиндром. Короче говоря, он должен решить, какие операции контроллера нужно выполнить и в какой последовательности.

После передачи команды контроллеру драйвер должен решить, блокировать ли себя до окончания заданной операции или нет. Если операция занимает значительное время, как при печати некоторого блока данных, то драйвер блокируется до тех пор, пока операция не завершится, и обработчик прерывания не разблокирует его. Если команда ввода-вывода выполняется быстро (например, прокрутка экрана), то драйвер ожидает ее завершения без блокирования.

Независимый от устройств слой операционной системы

Большая часть программного обеспечения ввода-вывода является независимой от устройств. Точная граница между драйверами и независимыми от устройств программами определяется системой, так как некоторые функции, которые могли бы быть реализованы независимым способом, в действительности выполнены в виде драйверов для повышения эффективности или по другим причинам.

Типичными функциями для независимого от устройств слоя являются:

- обеспечение общего интерфейса к драйверам устройств,
- именование устройств,
- защита устройств,
- обеспечение независимого размера блока,
- буферизация,
- распределение памяти на блок-ориентированных устройствах,
- распределение и освобождение выделенных устройств,
- уведомление об ошибках.

Остановимся на некоторых функциях данного перечня. Верхним слоям программного обеспечения не удобно работать с блоками разной величины, поэтому данный слой обеспечивает единый размер блока, например, за счет объединения нескольких различных блоков в единый логический блок. В связи с этим верхние уровни имеют дело с абстрактными устройствами, которые используют единый размер логического блока независимо от размера физического сектора.

При создании файла или заполнении его новыми данными необходимо выделить ему новые блоки. Для этого ОС должна вести список или битовую карту свободных блоков диска. На основании информации о наличии свободного места на диске может быть разработан алгоритм поиска свободного блока, независимый от устройства и реализуемый программным слоем, находящимся выше слоя драйверов.

2.1.3 Результаты и выводы:

Контрольные вопросы:

1. На какие типы делятся периферийные устройства?
2. Что такое ввод и вывод?
3. Что такое СУВВ и какова ее основная задача?
4. Какие действия могут выполняться в отношении ПУ?
5. Охарактеризуйте прямой метод управления ПУ.
6. Охарактеризуйте косвенный метод управления ПУ.
7. Что такое канал управления вводом-выводом?
8. Какие средства могут использоваться для синхронизации параллельной работы ЦП и канала? Кратко опишите принцип работы этих средств.
9. Какие функции выполняет супервизор ввода-вывода?
10. Что из себя представляет буфер, и для чего он используется?

2.2 Практическое занятие №3-4 (4 часа).

Тема: «Минимизация логических функций. Выполнение операций в двоичном коде**»**

2.2.1 Задание для работы:

1) Архитектура и структура компьютера
2) Состав и назначение основных устройств ЭВМ

2.2.2 Краткое описание проводимого занятия:

1) Создание и завершение процессов.

Создание процессов

Когда операционная система собирается добавить новый процесс к тем, которые уже состоят на учете, она создает структуры данных, использующиеся при управлении этим процессом (как описано в разделе 3.2), и размещает его адресное пространство в основной памяти. С помощью этих действий и создается новый процесс.

К созданию процесса могут привести четыре события, перечисленные в табл. 3.1. В среде пакетной обработки процесс создается в ответ на поступление задания; в интерактивной среде процесс создается при попытке нового пользователя войти в систему. В обоих случаях ответственность за создание нового процесса лежит на операционной системе. Кроме того, операционная система может создавать процесс по требованию приложения. Например, если пользователь отправляет запрос на распечатку файла, операционная система может создать процесс, управляющий печатью. Затем процесс, производивший запрос, может продолжить свою работу, независимо от того, сколько времени понадобится для печати.

Традиционно операционная система создает все процессы незаметно для пользователя или приложения; такой способ принят во многих современных операционных системах. Однако иногда требуется, чтобы один процесс мог послужить причиной создания другого процесса. Например, процесс приложения может сгенерировать другой процесс, который будет получать данные от первого процесса и приводить их к виду, удобному для дальнейшего анализа. Новый процесс будет работать параллельно с приложением и время от времени активизироваться для получения новых данных. Такая организация может быть очень полезна для структурирования приложений. В качестве другого примера можно привести ситуацию, в которой процесс-сервер (например, сервер печати или файловый сервер) может генерировать новый процесс для каждого обрабатываемого им запроса. Создание операционной системой процесса по явному запросу другого процесса называется порождением процесса (processspawning).

Когда один процесс порождает другой, то порождающий процесс называется родительским, или предком (parent), а порождаемый процесс — дочерним, или потомком

(child). Обычно "родственные" процессы обмениваются между собой информацией и взаимодействуют друг с другом. Организация такого взаимодействия является достаточно трудной задачей для программиста (эта тема рассматривается в главе 5, "Параллельные вычисления: взаимоисключения и многозадачность").

Завершение процессов

В табл. 3.2 перечислены типичные причины завершения процессов. В любой компьютерной системе должны быть средства, позволяющие определить, закончилось выполнение процесса или нет. Пакетное задание должно включать в себя команду типа Halt (останов) или какой-то явный вызов службы операционной системы, приводящий к завершению процесса. В первом случае генерируется прерывание для извещения операционной системы о завершении процесса. Например, в системе с разделением времени процесс пользователя должен быть завершен, когда пользователь выходит из системы или выключает терминал. На персональном компьютере или рабочей станции пользователь может выйти из приложения (например, закрыть программу обработки текста или электронную таблицу). Все эти действия в конечном счете приведут к тому, что будет вызвана служба операционной системы, завершающая процесс.

Наконец, в некоторых операционных системах процесс может быть завершен процессом, который его породил, а также при завершении самого родительского процесса.

2) Структуры управления процессами.

Важнейшей частью операционной системы, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами. Процесс (или по-другому, задача) - абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

Состояние процессов

В многозадачной (многопроцессной) системе процесс может находиться в одном из трех основных состояний:

ВЫПОЛНЕНИЕ - активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;

ОЖИДАНИЕ - пассивное состояние процесса, процесс заблокирован, он не может выполняться по своим внутренним причинам, он ждет осуществления некоторого события, например, завершения операции ввода-вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса;

ГОТОВНОСТЬ - также пассивное состояние процесса, но в этом случае процесс заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

В ходе жизненного цикла каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования процессов, реализуемым в данной операционной системе.

В состоянии ВЫПОЛНЕНИЕ в однопроцессорной системе может находиться только один процесс, а в каждом из состояний ОЖИДАНИЕ и ГОТОВНОСТЬ - несколько процессов, эти процессы образуют очереди соответственно ожидающих и готовых процессов. Жизненный цикл процесса начинается с состояния ГОТОВНОСТЬ, когда процесс готов к выполнению и ждет своей очереди. При активизации процесс переходит в состояние ВЫПОЛНЕНИЕ и находится в нем до тех пор, пока либо он сам освободит процессор, перейдя в состояние ОЖИДАНИЯ какого-нибудь события, либо будет

насильно "вытеснен" из процессора, например, вследствие исчерпания отведенного данному процессу кванта процессорного времени. В последнем случае процесс возвращается в состояние ГОТОВНОСТЬ. В это же состояние процесс переходит из состояния ОЖИДАНИЕ, после того, как ожидаемое событие произойдет.

3) Идентификатор и дескриптор процесса.

На протяжении существования процесса его выполнение может быть многократно прервано и продолжено. Для того, чтобы возобновить выполнение процесса, необходимо восстановить состояние его операционной среды. Состояние операционной среды отображается состоянием регистров и программного счетчика, режимом работы процессора, указателями на открытые файлы, информацией о незавершенных операциях ввода-вывода, кодами ошибок выполняемых данным процессом системных вызовов и т.д. Эта информация называется контекстом процесса.

Кроме этого, операционной системе для реализации планирования процессов требуется дополнительная информация: идентификатор процесса, состояние процесса, данные о степени привилегированности процесса, место нахождения кодового сегмента и другая информация. В некоторых ОС (например, в ОС UNIX) информацию такого рода, используемую ОС для планирования процессов, называют дескриптором процесса.

Дескриптор процесса по сравнению с контекстом содержит более оперативную информацию, которая должна быть легко доступна подсистеме планирования процессов. Контекст процесса содержит менее актуальную информацию и используется операционной системой только после того, как принято решение о возобновлении прерванного процесса.

Очереди процессов представляют собой дескрипторы отдельных процессов, объединенные в списки. Таким образом, каждый дескриптор, кроме всего прочего, содержит по крайней мере один указатель на другой дескриптор, соседствующий с ним в очереди. Такая организация очередей позволяет легко их переупорядочивать, включать и исключать процессы, переводить процессы из одного состояния в другое.

Программный код только тогда начнет выполняться, когда для него операционной системой будет создан процесс. Создать процесс - это значит:

- создать информационные структуры, описывающие данный процесс, то есть его дескриптор и контекст;
- включить дескриптор нового процесса в очередь готовых процессов;
- загрузить кодовый сегмент процесса в оперативную память или в область свопинга.

2.2.3 Результаты и выводы:

Контрольные вопросы:

1) Что такое процесс?
2) В каких случаях ОС создает и завершает процессы?
3) Назовите все состояния процессов
4) Что такое идентификатор процесса?
5) Что такое дескриптор процесса?
6) Описать подробно создание процесса

2.3 Практическое занятие №5-6 (4 часа).

Тема: «Построение логических схем. Комбинированные узлы**»**

2.3.1 Задание для работы:

1) Организация памяти. Микросхемы памяти
2) Процессор. Функции, параметры, структура процессора

2.3.2 Краткое описание проводимого занятия:

1) Выполнение кода операционной системы.

В главе, "Обзор операционных систем", были отмечены два занимательных факта, касающиеся операционных систем.

- Операционная система работает точно так же, как и обычная программа; т.е. она тоже является программой, которая выполняется процессором.
- Операционная система часто передает управление другим программам; возврат управления операционной системе зависит от процессора.

Если операционная система представляет собой обычный набор программ и если она выполняется процессором точно так же, как и любая другая программа, то является ли операционная система процессом? Если это так, то как осуществляется управление этим процессом? Размышления над этими интересными вопросами стали причиной появления различных подходов к разработке операционных систем. На рис. 3.12 проиллюстрированы подходы, реализованные в различных операционных системах.

Ядро вне процессов

Один из традиционных подходов, который применялся во многих ранних операционных системах, состоит в том, чтобы выполнять ядро операционной системы вне всяких процессов (рис. 3.12,а). При таком подходе прерывание выполняющегося в данное время процесса или вызов управляющей программы приводит к сохранению контекста данного процесса и передаче управления ядру. Операционная система имеет свою собственную область памяти и свой собственный системный стек, который используется для управления вызовами процедур и возвратами из них. Операционная система может выполнить все необходимые функции и восстановить контекст прерванного процесса, после чего выполнение этого процесса будет продолжено. После завершения сохранения контекста данного процесса операционная система может также перейти к планированию и диспетчеризации другого процесса. Случится это или нет — зависит от того, что именно послужило причиной прерывания, и от ряда других обстоятельств.

В любом случае основным моментом такой схемы является то, что концепция процесса рассматривается в ней лишь применительно к пользовательским программам. Код операционной системы выполняется как некий отдельный объект, работающий в привилегированном режиме.

Выполнение в составе пользовательских процессов

На небольших машинах (персональных компьютерах, рабочих станциях) часто применяется альтернативный подход, при котором почти все программы операционной системы выполняются в контексте пользовательского процесса. Разработчики таких систем придерживаются той точки зрения, что операционная система — это в основном набор процедур, которые вызываются для выяснения различных функций пользовательского процесса. Этот подход проиллюстрирован на рис. 3.12,6. Каждый процесс, принятый операционной системой на обработку, включает в себя не только блоки, изображенные на рис. 3.10, но и области кода, данных и стека программ ядра.

На рис. 3.13 показана типичная схема структуры образа процесса, принятая в такой стратегии. Для управления вызовом системных процедур, работающих в режиме ядра, и возврата из них используется отдельный стек ядра. Код и данные операционной системы находятся в совместно используемом адресном пространстве и доступны для использования всеми пользовательскими процессами.

При прерывании, системном прерывании или вызове управляющей программы процессор переходит в режим ядра, а управление передается операционной системе. Чтобы это произошло, сохраняется контекст процесса, и происходит переключение режима с передачей управления процедуре операционной системы. Однако выполнение текущего пользовательского процесса продолжается. Таким образом, переключения процесса не происходит, переключается только режим работы процессора в рамках одного и того же процесса.

Если операционная система по завершении своей задачи придет к заключению, что следует продолжить текущий процесс, то с переключением режима процессора в предыдущее состояние возобновляется выполнение прерванной программы в рамках текущего процесса. Одно из основных преимуществ такого подхода состоит в следующем: если пользовательская программа прервалась, чтобы выполнить некоторую процедуру операционной системы, а затем возобновила свою работу, нам удается избежать двух излишних переключений процессов. Если же операционная система определит, что нужно переключить процесс, а не продолжать выполнение предыдущей программы, то управление переходит к процедуре, выполняющей переключение процессов. В зависимости от архитектуры операционной системы эта процедура может выполняться либо в составе текущего процесса, либо в составе некоторого другого процесса. В любом случае в какой-то момент текущий процесс нужно будет переключить, сняв его с выполнения, а в состояние выполнения перевести другой процесс. С точки зрения логики эту фазу удобнее всего рассматривать как нечто, происходящее вне всех процессов.

В некотором отношении такая точка зрения на операционную систему является довольно интересной. Выполняющийся процесс в определенный момент времени сам сохраняет информацию о своем состоянии, выбирает для выполнения другой процесс, находящийся в состоянии готовности, и передает ему управление. Причина того, что в такой ситуации не наступает хаос и произвол, заключается в том, что критичный код представляет собой не код пользовательской программы, а совместно используемый код операционной системы, выполняющийся в контексте процесса. В силу наличия пользовательского режима и режима ядра пользователь не может вмешиваться в работу системных процедур, хотя они и выполняются в среде пользовательского процесса. Это еще раз напоминает нам об отличиях концепций процесса и программы и о том, что между ними нельзя ставить знак равенства. В ходе процесса могут выполняться и пользовательские программы, и программы операционной системы; с другой стороны, программы операционной системы, выполняемые в разных пользовательских процессах, являются идентичными.

2) Управление процессами в операционных системах Windows.

Процесс (process) - это пользовательская программа при ее исполнении в компьютерной системе. Для выполнения процесса требуется ряд ресурсов, включая время процессора, память, файлы, устройства ввода-вывода, сетевые устройства и др.

В классической схеме UNIX, при создании процесса для него создается новое пространство виртуальной памяти, т.е. таблица страниц для отображения виртуальных адресов в физические, своя для каждого нового процесса. При этом расходуются значительные ресурсы. Если учесть, что в UNIX каждая команда пользователя (например, ls – вывод содержимого текущей директории) запускается как отдельный процесс, то становится понятным, насколько "дорога" операция создания процесса в классическом смысле. Поэтому еще в 1980-х гг. появилась концепция облегченного процесса (lightweightprocess) – выполняемого в том же пространстве виртуальной памяти, что и процесс-родитель. При создании нового облегченного процесса ОС создает для него только стек – системный резидентный массив в памяти, предназначенный для поддержки выполнения процедур процесса и хранящий их локальные данные и связующую информацию между ними.

ОС отвечает за следующие действия, связанные с управлением процессами:

Создание и удаление процессов. При создании процесса необходимо создать в памяти соответствующие системные структуры (таблицу страниц, стек и др.). При удалении процесса память, занимаемая ими, освобождается, а также выполняется закрытие всех файлов и освобождение всех других ресурсов, которые использовал процесс, если последний не сделал этого явно.

Приостановка и возобновление процессов. Выполнение процесса приостанавливается при выполнении синхронного ввода-вывода, а также системного вызова или команды (типа suspend). Сразу отметим, что использовать подобные операции явной приостановки процессов следует с осторожностью, так как приостанавливаемый процесс может находиться в своей критической секции – выполнять обработку общего ресурса, к которому каждому процессу предоставляется монопольный доступ, так что при его приостановке возникает ситуация тупика (deadlock) – приостановленный процесс не может освободить ресурс, а конкурирующий процесс не может его получить. При приостановке процесса ОС сохраняет состояние его выполнения, а при возобновлении – восстанавливает.

Синхронизация процессов. Процессы работают параллельно и при этом конкурируют за общие ресурсы, а также должны в некоторые моменты вычислений ожидать наступления некоторых событий. Для предотвращения возможных конфликтов и несогласованностей, например, racecondition - несогласованного доступа к общим данным, при котором один процесс читает старые данные, а другой их в этот же момент обновляет, - ОС предоставляет средства синхронизации (например, семафоры и мониторы, рассмотренные в следующем разделе).

Взаимодействие процессов. При своей параллельной работе процессам необходимо взаимодействие, с целью согласованного решения различных частей одной и той же задачи. Процессы могут взаимодействовать с помощью передачи сообщений друг другу, а также с помощью так называемых условных переменных и рандеву (все эти виды взаимодействия рассмотрены позже). ОС предоставляет все эти средства, в виде системных вызовов, для организации адекватного и удобного взаимодействия процессов.

2..3 Результаты и выводы:

Контрольные вопросы:

1) Что такое процесс?
2) Особенность подхода "ядро вне процессов"
3) Преимущества выполнения кода ОС в составе пользовательских процессов
4) За какие действия усправления процессами отвечает ОС?
5) Что такое синхронизация процессов?

2.4 Практическое занятие №7-8 (4 часа).

Тема: «Узлы с памятью»

2.1 Задание для работы:

1) Введение в язык Ассемблера.
2) Язык Ассемблера. Основные команды.

2.4.2 Краткое описание проводимого занятия:

1) Потоки на пользовательском уровне и на уровне ядра.

Обычно выделяют две общие категории потоков: потоки на уровне пользователя (user-levelthreads — ULT) и потоки на уровне ядра (kernel-levelthreads — KLT). Потоки второго типа в литературе иногда называются потоками, поддерживаемыми ядром, или облегченными процессами.

Потоки на уровне пользователя

В программе, полностью состоящей из ULT-потоков, все действия по управлению потоками выполняются самим приложением; ядро, по сути, и не подозревает о существовании потоков. На рис. 4.6,а проиллюстрирован подход, при котором используются только потоки на уровне пользователя. Чтобы приложение было многопоточным, его следует создавать с применением специальной библиотеки,

представляющей собой пакет программ для работы с потоками на уровне ядра. Такая библиотека для работы с потоками содержит код, с помощью которого можно создавать и удалять потоки, производить обмен сообщениями и данными между потоками, планировать их выполнение, а также сохранять и восстанавливать их контекст.

По умолчанию приложение в начале своей работы состоит из одного потока и его выполнение начинается как выполнение этого потока. Такое приложение вместе с составляющим его потоком размещается в едином процессе, который управляется ядром. Выполняющееся приложение в любой момент времени может породить новый поток, который будет выполняться в пределах того же процесса. Новый поток создается с помощью вызова специальной подпрограммы из библиотеки, предназначенной для работы с потоками. Управление к этой подпрограмме переходит в результате вызова процедуры. Библиотека потоков создает структуру данных для нового потока, а потом передает управление одному из готовых к выполнению потоков данного процесса, руководствуясь некоторым алгоритмом планирования. Когда управление переходит к библиотечной подпрограмме, контекст текущего потока сохраняется, а когда управление возвращается к потоку, его контекст восстанавливается. Этот контекст в основном состоит из содержимого пользовательских регистров, счетчика команд и указателей стека.

Все описанные в предыдущих абзацах события происходят в пользовательском пространстве в рамках одного процесса. Ядро не подозревает об этой деятельности. Оно продолжает осуществлять планирование процесса как единого целого и приписывать ему единое состояние выполнения (состояние готовности, состояние выполняющегося процесса, состояние блокировки и т.д.). Приведенные ниже примеры должны прояснить взаимосвязь между планированием потоков и планированием процессов. Предположим, что выполняется поток 2, входящий в процесс В (см. рис. 4.7). Состояния этого процесса и составляющих его потоков на пользовательском уровне показаны на рис. 4.7,а. Впоследствии может произойти одно из следующих событий.

Приложение, в котором выполняется поток 2, может произвести системный вызов, например запрос ввода-вывода, который блокирует процесс В. В результате этого вызова управление перейдет к ядру. Ядро вызывает процедуру ввода-вывода, переводит процесс В в состояние блокировки и передает управление другому процессу. Тем временем поток 2 процесса В все еще находится в состоянии выполнения в соответствии со структурой данных, поддерживаемой библиотекой потоков. Важно отметить, что поток 2 не выполняется в том смысле, что он работает с процессором; однако библиотека потоков воспринимает его как выполняющийся. Соответствующие диаграммы состояний показаны на рис. 4.7,6.

В результате прерывания по таймеру управление может перейти к ядру; ядро определяет, что интервал времени, отведенный выполняющемуся в данный момент процессу В, истек. Ядро переводит процесс В в состояние готовности и передает управление другому процессу. В это время, согласно структуре данных, которая поддерживается библиотекой потоков, поток 2 процесса В по-прежнему будет находиться в состоянии выполнения. Соответствующие диаграммы состояний показаны на рис. 4.7,в.

Поток 2 достигает точки выполнения, когда ему требуется, чтобы поток 1 процесса В выполнил некоторое действие. Он переходит в заблокированное состояние, а поток 1 — из состояния готовности в состояние выполнения. Сам процесс остается в состоянии выполнения. Соответствующие диаграммы состояний показаны на рис. 4.7,г.

В случаях 1 и 2 (см. рис. 4.7,6 и в) при возврате управления процессу В возобновляется выполнение потока 2. Заметим также, что процесс, в котором выполняется код из библиотеки потоков, может быть прерван либо из-за того, что закончится отведенный ему интервал времени, либо из-за наличия процесса с более высоким приоритетом. Когда возобновится выполнение прерванного процесса, оно продолжится работой процедуры из библиотеки потоков, которая завершит переключение потоков и передаст управление новому потоку процесса.

Использование потоков на пользовательском уровне обладает некоторыми преимуществами перед использованием потоков на уровне ядра. К этим преимуществам относятся следующие:

Переключение потоков не включает в себя переход в режим ядра, так как структуры данных по управлению потоками находятся в адресном пространстве одного и того же процесса. Поэтому для управления потоками процессу не нужно переключаться в режим ядра. Благодаря этому обстоятельству удается избежать накладных расходов, связанных с двумя переключениями режимов (пользовательского режима в режим ядра и обратно).

Планирование производится в зависимости от специфики приложения. Для одних приложений может лучше подойти простой алгоритм планирования по круговому алгоритму, а для других — алгоритм планирования, основанный на использовании приоритета. Алгоритм планирования может подбираться для конкретного приложения, причем это не повлияет на алгоритм планирования, заложенный в операционной системе.

Использование потоков на пользовательском уровне применимо для любой операционной системы. Для их поддержки в ядро системы не потребуется вносить никаких изменений. Библиотека потоков представляет собой набор утилит, работающих на уровне приложения и совместно используемых всеми приложениями.

Использование потоков на пользовательском уровне обладает двумя явными недостатками по сравнению с использованием потоков на уровне ядра.

В типичной операционной системе многие системные вызовы являются блокирующими. Когда в потоке, работающем на пользовательском уровне, выполняется системный вызов, блокируется не только данный поток, но и все потоки того процесса, к которому он относится.

В стратегии с наличием потоков только на пользовательском уровне приложение не может воспользоваться преимуществами многопроцессорной системы, так как ядро закрепляет за каждым процессом только один процессор. Поэтому несколько потоков одного и того же процесса не могут выполняться одновременно. В сущности, у нас получается многозадачность на уровне приложения в рамках одного процесса. Несмотря на то, что даже такая многозадачность может привести к значительному увеличению скорости работы приложения, имеются приложения, которые работали бы гораздо лучше, если бы различные части их кода могли выполняться одновременно.

Эти две проблемы разрешимы. Например, их можно преодолеть, если писать приложение не в виде нескольких потоков, а в виде нескольких процессов. Однако при таком подходе основные преимущества потоков сводятся на нет: каждое переключение становится не переключением потоков, а переключением процессов, что приведет к значительно большим накладным затратам.

Другим методом преодоления проблемы блокирования является использование преобразования блокирующего системного вызова в неблокирующий. Например, вместо непосредственного вызова системной процедуры ввода-вывода поток вызывает подпрограмму-оболочку, которая производит ввод-вывод на уровне приложения. В этой программе содержится код, который проверяет, занято ли устройство ввода-вывода. Если оно занято, поток передает управление другому потоку (что происходит с помощью библиотеки потоков). Когда наш поток вновь получает управление, он повторно осуществляет проверку занятости устройства ввода-вывода.

Потоки на уровне ядра

В программе, работа которой полностью основана на потоках, работающих на уровне ядра, все действия по управлению потоками выполняются ядром. В области приложений отсутствует код, предназначенный для управления потоками. Вместо него используется интерфейс прикладного программирования (applicationprogramminginterface — API) средств ядра, управляющих потоками. Примерами такого подхода являются операционные системы OS/2, Linux и W2K.

На рис. 4.6,6" проиллюстрирована стратегия использования потоков на уровне ядра. Любое приложение при этом можно запрограммировать как многопоточное; все потоки приложения поддерживаются в рамках единого процесса. Ядро поддерживает информацию контекста процесса как единого целого, а также контекстов каждого отдельного потока процесса. Планирование выполняется ядром исходя из состояния потоков. С помощью такого подхода удается избавиться от двух упомянутых ранее основных недостатков потоков пользовательского уровня. Во-первых, ядро может одновременно осуществлять планирование работы нескольких потоков одного и того же процесса на нескольких процессорах. Во-вторых, при блокировке одного из потоков процесса ядро может выбрать для выполнения другой поток этого же процесса. Еще одним преимуществом такого подхода является то, что сами процедуры ядра могут быть многопоточными.

Основным недостатком подхода с использованием потоков на уровне ядра по сравнению с использованием потоков на пользовательском уровне является то, что для передачи управления от одного потока другому в рамках одного и того же процесса приходится переключаться в режим ядра. Результаты исследований, проведенных на однопроцессорной машине VAX под управлением UNIX-подобной операционной системы, представленные в табл. 4.1, иллюстрируют различие между этими двумя подходами. Сравнивалось время выполнения таких двух задач, как (1) нулевое ветвление (NullFork) — время, затраченное на создание, планирование и выполнение процесса/потока, состоящего только из нулевой процедуры (измеряются только накладные расходы, связанные с ветвлением процесса/потока), и (2) ожидание сигнала (Signal-Wait) — время, затраченное на передачу сигнала от одного процесса/потока другому процессу/потоку, находящемуся в состоянии ожидания (накладные расходы на синхронизацию двух процессов/потоков). Чтобы было легче сравнивать полученные значения, заметим, что вызов процедуры на машине VAX, используемой в этом исследовании, длится 7 us, а системное прерывание — 17 us. Мы видим, что различие во времени выполнения потоков на уровне ядра и потоков на пользовательском уровне более чем на порядок превосходит по величине различие во времени выполнения потоков на уровне ядра и процессов.

Таким образом, создается впечатление, что как применение многопоточности на уровне ядра дает выигрыш по сравнению с процессами, так и многопоточность на пользовательском уровне дает выигрыш по сравнению с многопоточностью на пользовательском уровне. Однако на деле возможность этого дополнительного выигрыша зависит от характера приложений. Если для большинства переключений потоков приложения необходим доступ к ядру, то схема с потоками на пользовательском уровне может работать ненамного лучше, чем схема с потоками на уровне ядра.

2) Управление процессами и потоками в Windows.

Одной из основных подсистем любой современной мультипрограммной ОС, непосредственно влияющей на функционирование компьютера, является подсистема управления процессами и потоками. Основные функции этой подсистемы:

- создание процессов и потоков;
- обеспечение процессов и потоков необходимыми ресурсами;
- изоляция процессов;
- планирование выполнения процессов и потоков (вообще, следует говорить и о планировании заданий);
- диспетчеризация потоков;
- организация межпроцессного взаимодействия;
- синхронизация процессов и потоков;
- завершение и уничтожение процессов и потоков.

К созданию процесса приводят пять основных событий:

- инициализация ОС (загрузка);
- выполнение запроса работающего процесса на создание процесса;
- запрос пользователя на создание процесса, например, при входе в систему в интерактивном режиме;
- инициирование пакетного задания;
- создание операционной системой процесса, необходимого для работы каких-либо служб.

Обычно при загрузке ОС создаются несколько процессов. Некоторые из них являются высокоприоритетными процессами, обеспечивающими взаимодействие с пользователями и выполняющими заданную работу. Остальные процессы являются фоновыми, они не связаны с конкретными пользователями, но выполняют особые функции – например, связанные с электронной почтой, Web-страницами, выводом на печать, передачей файлов по сети, периодическим запуском программ (например, дефрагментации дисков) и т.д. Фоновые процессы называют демонами.

Новый процесс может быть создан по запросу текущего процесса. Создание новых процессов полезно в тех случаях, когда выполняемую задачу проще всего сформировать как набор связанных, но, тем не менее, независимых взаимодействующих процессов. В интерактивных системах пользователь может запустить программу, набрав на клавиатуре команду или дважды щелкнув на значке программы. В обоих случаях создается новый процесс и запуск в нем программы. В системах пакетной обработки на мэйнфреймах пользователи посылают задание (возможно, с использованием удаленного доступа), а ОС создает новый процесс и запускает следующее задание из очереди, когда освобождаются необходимые ресурсы.

С технической точки зрения во всех перечисленных случаях новый процесс формируется одинаково: текущий процесс выполняет системный запрос на создание нового процесса. Подсистема управления процессами и потоками отвечает за обеспечение процессов необходимыми ресурсами. ОС поддерживает в памяти специальные информационные структуры, в которые записывает, какие ресурсы выделены каждому процессу. Она может назначить процессу ресурсы в единоличное пользование или совместное пользование с другими процессами. Некоторые из ресурсов выделяются процессу при его создании, а некоторые – динамически по запросам во время выполнения. Ресурсы могут быть выделены процессу на все время его жизни или только на определенный период. При выполнении этих функций подсистема управления процессами взаимодействует с другими подсистемами ОС, ответственными за управление ресурсами, такими как подсистема управления памятью, подсистема ввода-вывода, файловая система.

Для того чтобы процессы не могли вмешаться в распределение ресурсов, а также не могли повредить коды и данные друг друга, важнейшей задачей ОС является изоляция одного процесса от другого. Для этого операционная система обеспечивает каждый процесс отдельным виртуальным адресным пространством, так что ни один процесс не может получить прямого доступа к командам и данным другого процесса.

В ОС, где существуют процессы и потоки, процесс рассматривается как заявка на потребление всех видов ресурсов, кроме одного – процессорного времени. Этот важнейший ресурс распределяется операционной системой между другими единицами работы – потоками, которые и получили свое название благодаря тому, что они представляют собой последовательности (потоки выполнения) команд. Переход от выполнения одного потока к другому осуществляется в результате планирования и диспетчеризации. Работа по определению момента, в который необходимо прервать выполнение текущего потока, и потока, которому следует предоставить возможность выполняться, называется планированием. Планирование потоков осуществляется на основе информации, хранящейся в описателях процессов и потоков. При планировании

принимается во внимание приоритет потоков, время их ожидания в очереди, накопленное время выполнения, интенсивность обращения к вводу-выводу и другие факторы.

Диспетчеризация заключается в реализации найденного в результате планирования решения, т.е. в переключении процессора с одного потока на другой. Диспетчеризация проходит в три этапа:

- сохранение контекста текущего потока;
- загрузка контекста потока, выбранного в результате планирования;
- запуск нового потока на выполнение.

Когда в системе одновременно выполняется несколько независимых задач, возникают дополнительные проблемы. Хотя потоки возникают и выполняются синхронно, у них может возникнуть необходимость во взаимодействии, например, при обмене данными. Для общения друг с другом процессы и потоки могут использовать широкий спектр возможностей: каналы (в UNIX), почтовые ящики (Windows), вызов удаленной процедуры, сокеты (в Windows соединяют процессы на разных машинах). Согласование скоростей потоков также очень важно для предотвращения эффекта "гонок" (когда несколько потоков пытаются изменить один и тот же файл), взаимных блокировок и других коллизий, которые возникают при совместном использовании ресурсов.

Синхронизация потоков является одной из важнейших функций подсистемы управления процессами и потоками. Современные операционные системы предоставляют множество механизмов синхронизации, включая семафоры, мьютексы, критические области и события. Все эти механизмы работают с потоками, а не с процессами. Поэтому когда поток блокируется на семафоре, другие потоки этого процесса могут продолжать работу.

Каждый раз, когда процесс завершается, – а это происходит благодаря одному из следующих событий: обычный выход, выход по ошибке, выход по неисправимой ошибке, уничтожение другим процессом – ОС предпринимает шаги, чтобы "зачистить следы" его пребывания в системе. Подсистема управления процессами закрывает все файлы, с которыми работал процесс, освобождает области оперативной памяти, отведенные под коды, данные и системные информационные структуры процесса. Выполняется коррекция всевозможных очередей ОС и список ресурсов, в которых имелись ссылки на завершаемый процесс.

Как уже отмечалось, чтобы поддержать мультипрограммирование, ОС должна оформить для себя те внутренние единицы работы, между которыми будет разделяться процессор и другие ресурсы компьютера. Возникает вопрос: в чем принципиальное отличие этих единиц работы, какой эффект мультипрограммирования можно получить от их применения и в каких случаях эти единицы работ операционной системы следует создавать?

Очевидно, что любая работа вычислительной системы заключается в выполнении некоторой программы. Поэтому и с процессом, и с потоком связывается определенный программный код, который оформляется в виде исполняемого модуля. В простейшем случае процесс состоит из одного потока, и в некоторых современных ОС сохранилось такое положение. Мультипрограммирование в таких ОС осуществляется на уровне процессов. При необходимости взаимодействия процессы обращаются к операционной системе, которая, выполняя функции посредника, предоставляет им средства межпроцессной связи – каналы, почтовые акции, разделяемые секции памяти и др.

Однако в системах, в которых отсутствует понятие потока, возникают проблемы при организации параллельных вычислений в рамках процесса. А такая необходимость может возникать. Дело в том, что отдельный процесс никогда не может быть выполнен быстрее, чем в однопрограммном режиме. Однако приложение, выполняемое в рамках одного процесса, может обладать внутренним параллелизмом, который, в принципе, мог бы ускорить его решение. Если, например, в программе предусмотрено обращение к

внешнему устройству, то на время этой операции можно не блокировать выполнение всего процесса, а продолжить вычисления по другой ветви программы.

Параллельное выполнение нескольких работ в рамках одного интерактивного приложения повышает эффективность работы пользователя. Так, при работе с текстовым редактором желательно иметь возможность совмещения набора нового текста с такими продолжительными операциями, как переформатирование значительной части текста, сохранение его на локальном или удаленном диске.

Нетрудно представить будущую версию компилятора, способную автоматически компилировать файлы исходного кода в паузах, возникающих при наборе текста программы. Тогда предупреждения и сообщения об ошибках появлялись бы в режиме реального времени, и пользователь тут же видел бы, в чем он ошибся. Современные электронные таблицы пересчитывают данные в фоновом режиме, как только пользователь что-либо изменил. Текстовые процессоры разбивают текст на страницы, проверяют его на орфографические и грамматические ошибки, печатают в фоновом режиме, сохраняют текст каждые несколько минут и т.д. Во всех этих случаях потоки используются как средство распараллеливания вычислений.

Эти задачи можно было бы возложить на программиста, который должен был бы написать программу-диспетчер, реализующую параллелизм в рамках одного процесса. Однако это весьма сложно, да и сама программа получилась бы весьма запутанной и сложной в отладке.

Другим решением является создание для одного приложения нескольких процессов для каждой из параллельных работ. Однако использование для создания процессов стандартных средств ОС не позволяет учесть тот факт, что процессы решают единую задачу и имеют много общего: работают с одними и теми же данными, используют один и тот же кодовый сегмент, имеют одни и те же права доступа к ресурсам вычислительной системы. А операционная система при таком подходе будет рассматривать эти процессы наравне со всеми остальными процессами и обеспечивать их изоляцию друг от друга. В данном случае это будет не только бесполезная, но и вредная работа, затрудняющая обмен данными между различными частями приложения. Кроме того, на создание каждого процесса ОС тратит определенные системные ресурсы, которые в данном случае неоправданно дублируются – каждому процессу выделяется собственное виртуальное адресное пространство, физическая память, закрепляются устройства ввода-вывода и т.п.

Из изложенного следует вывод, что операционной системе наряду с процессами нужен другой механизм распараллеливания вычислений, который учитывал бы тесные связи между отдельными ветвями вычислений одного и того же приложения. Для этих целей современные ОС предлагают механизм многопоточной обработки (multithreading).

Понятию "поток" соответствует последовательный переход процессора от одной команды к другой. Процессу ОС назначают адресное пространство и набор ресурсов, которые совместно используются всеми его потоками. В отличие от процессов, которые принадлежат, вообще говоря, конкурирующим приложениям, все потоки одного процесса всегда принадлежат одному приложению, поэтому ОС изолирует потоки в гораздо меньшей степени, чем процессы в традиционной мультипрограммной системе. Все потоки одного процесса используют общие файлы, таймеры, устройства, одну и ту же область оперативной памяти, одно и то же адресное пространство.

Это означает, что они разделяют одни и те же глобальные переменные. Поскольку каждый поток может иметь доступ к любому виртуальному адресу, один поток может задействовать стек другого потока. Между потоками одного процесса нет полной защиты, во-первых, потому что это невозможно, а во-вторых, потому что не нужно. Чтобы организовать взаимодействие и обмен данными, потокам не требуется обращаться к ОС, им достаточно использовать общую память – один поток записывает данные, а другой читает их. С другой стороны, потоки разных процессов по-прежнему хорошо защищены друг от друга.

Таким образом, мультипрограммирование более эффективно на уровне потоков, а не процессов. Еще больший эффект многопоточной обработки достигается в мультипроцессорных системах, в которых потоки могут выполняться на разных процессорах действительно параллельно.

2.4.3 Результаты и выводы:

Контрольные вопросы:

1) Что такое семафор, в чем его функция?
2) Какие 3 операции могут быть произведены над семафором?
3) Что такое очередь?
4) Что такое монитор, его функции?
5) Характеристики монитора
6) Какие механизмы для синхронизации и связи между процессами вы знаете?
7) Что такое каналы?
8) Работа разделяемой памяти
9) Что такое сигнал?

2.5 Практическое занятие №9-10 (4 часа).

Тема: «Структуры запоминающих устройств ЭВМ. Структура ОЗУ»

2.5.1 Задание для работы:

1) Загрузочный сектор.
2) Файловая система FAT.

2.5.2 Краткое описание проводимого занятия:

Теперь мы вернемся к механизмам операционных систем и языков программирования, обеспечивающим параллельные вычисления. Этот раздел мы начнем с рассмотрения семафоров; следующие разделы будут посвящены мониторам и передаче сообщений.

Первой большой работой, посвященной вопросам параллельных вычислений, стала монография Дейкстры [DIJK65], который рассматривал разработку операционной системы как построение множества сотрудничающих последовательных процессов и создание эффективных и надежных механизмов поддержки этого сотрудничества. Эти же механизмы легко применяются и пользовательскими процессами — если процессор и операционная система делают их общедоступными.

Фундаментальный принцип заключается в том, что два или большее количество процессов могут сотрудничать посредством простых сигналов, так что в определенном месте процесс может приостановить работу до тех пор, пока не дождется соответствующего сигнала. Требования кооперации любой степени сложности могут быть удовлетворены соответствующей структурой сигналов. Для сигнализации используются специальные переменные, называющиеся семафорами. Для передачи сигнала через семафор s процесс выполняет примитив signal (s), а для получения сигнала— примитив wait(s). В последнем случае процесс приостанавливается до тех пор, пока не осуществится передача соответствующего сигнала.2

Для достижения желаемого эффекта мы можем рассматривать семафор как переменную, имеющую целое значение, над которой определены три операции.

- Семафор может быть инициализирован неотрицательным значением.
- Операция wait уменьшает значение семафора. Если это значение становится отрицательным, процесс, выполняющий операцию wait, блокируется.
- Операция signal увеличивает значение семафора. Если это значение не положительно, то заблокированный операцией wait процесс деблокируется.

Не имеется никаких иных способов получения информации о значении семафора или изменения его значения, кроме перечисленных.

В листинге 5.5 приведено более формальное определение примитивов семафоров. Предполагается, что примитивы wait и signalатомарны, т.е. они не могут быть прерваны, и каждая из подпрограмм может рассматриваться как единый шаг. Более ограниченная версия семафора, известная как бинарный семафор, представлена в листинге 5.6. Бинарный семафор может принимать только значения 0 или 1. В принципе реализация бинарного семафора должна быть более простой задачей; можно также показать, что все задачи, решаемые с применением обычных семафоров, могут быть решены и с использованием лишь бинарных семафоров

Для хранения процессов, ожидающих как обычные, так и бинарные семафоры, используется очередь. При этом возникает вопрос о порядке извлечения процессов из данной очереди. Наиболее корректный способ — использование принципа "первым вошел — первым вышел" (first-in-first-out — FIFO). При этом первым из очереди освобождается процесс, который был заблокирован дольше других. Семафор, использующий данный метод, называется сильным семафором (strongsemaphore). Семафор, порядок извлечения процессов из очереди которого не определен, называется слабым семафором (weaksemaphore). На рис. 5.2 (из [DENN84]) приведен пример работы сильного семафора. Здесь процессы A, B и C зависят от результатов работы процесса D. Изначально работает процесс A (®); процессы B, C и D находятся в списке активных процессов, ожидая своей очереди. Значение семафора равно 1, это указывает на то, что один из результатов работы процесса D имеется в наличии. Когда процесс A выполняет инструкцию wait, он тут же получает разрешение на дальнейшую работу и вновь становится в очередь на выполнение в списке активных процессов. Затем приступает к работе процесс B (©), который в конечном счете также выполняет инструкцию wait, в результате чего процесс приостанавливается, давая возможность приступить к работе процессу D (®). Когда процесс D завершает работу над получением нового результата, он выполняет инструкцию signal, которая позволяет процессу B перейти из списка приостановленных процессов в список активных (®). Процесс D присоединяется к списку активных процессов, и к выполнению приступает процесс C (©), но тут же приостанавливается при выполнении инструкции wait. Точно так же приостанавливается и выполнение процессов A и B, давая возможность процессу D приступить к работе (©). После того как получается новый результат процесса D, им выполняется инструкция signal, которая и переводит процесс C из списка приостановленных в список активных. Последующие циклы выполнения процесса D переведут в список активных процессы A и B.

Мониторы

Семафоры обеспечивают достаточно мощный и гибкий инструмент для осуществления взаимных исключений и координации процессов. Однако, как вы видели в листинге 5.8, создать корректно работающую программу с использованием семафоров не всегда легко. Сложность заключается в том, что операции wait и signal могут быть разбросаны по всей программе, и не всегда можно сразу отследить их воздействие на контролируемые ими семафоры.

Монитор представляет собой конструкцию языка программирования, которая обеспечивает функциональность, эквивалентную функциональности семафоров, но легче управляется. Впервые формальное определение концепции мониторов было дано в [HOAR74]. Мониторы реализованы во множестве языков программирования, включая такие, как ConcurrentPascal, Pascal-Plus, Modula-2, Modula-3 и Java. Мониторы также реализуются как программные библиотеки. Это позволяет использовать мониторы, блокирующие любые объекты. В частности, например, для связанного списка можно заблокировать все связанные списки одной блокировкой, либо иметь отдельные блокировки для каждого списка, а возможно — и для каждого элемента списка.

Рассмотрение мониторов мы начнем с версии Хоара (Hoare).

Мониторы с сигналами

Монитор представляет собой программный модуль, состоящий из инициализирующей последовательности, одной или нескольких процедур и локальных данных. Основными характеристиками монитора являются также.

- Локальные переменные монитора доступны только его процедурам; внешние процедуры доступа к локальным данным монитора не имеют.
- Процесс входит в монитор путем вызова одной из его процедур.
- В мониторе в определенный момент времени может выполняться только один процесс; любой другой процесс, вызвавший монитор, будет приостановлен в ожидании доступности монитора.

Первые две характеристики сразу заставляют нас вспомнить о объектах в объектно-ориентированном программировании. Фактически объектно-ориентированные операционные системы или языки программирования могут легко реализовать монитор как объект со специальными характеристиками.

Соблюдение условия выполнения только одного процесса в определенный момент времени позволяет монитору обеспечить взаимоисключения. Данные монитора доступны в этот момент только одному процессу, следовательно, защитить совместно используемые структуры данных можно, просто поместив их в монитор. Если данные в мониторе представляют некий ресурс, то монитор обеспечивает взаимоисключение при обращении к ресурсу.

Для широкого применения в параллельных вычислениях мониторы должны включать инструменты синхронизации. Предположим, например, что процесс использует монитор и, находясь в мониторе, должен быть приостановлен до выполнения некоторого условия. При этом нам требуется некий механизм, который не только приостанавливает процесс, но и освобождает монитор, позволяя войти в него другому процессу. Позже, когда условие окажется выполненным, а монитор доступным, приостановленный процесс сможет продолжить свою работу с того места, где он был приостановлен.

Монитор поддерживает синхронизацию при помощи переменных условия, располагающихся (и доступных) только в мониторе. Работать с этими переменными могут две функции.

cwait(c): приостанавливает выполнение вызывающего процесса по условию c. Монитор при этом доступен для использования другим процессом.

csignal(c): возобновляет выполнение некоторого процесса, приостановленного вызовом cwait с тем же условием. Если имеется несколько такихпроцессов, выбирается один из них; если таких процессов нет, функция неделает ничего.

Обратите внимание на то, что операции wait/signal монитора отличаются от соответствующих операций семафора. Если процесс в мониторе передает сигнал, но при этом нет ни одного ожидающего его процесса, то сигнал просто теряется.

На рис. 5.7 показана структура монитора. Хотя процесс может войти в монитор посредством вызова любой его процедуры, мы все же будем рассматривать монитор как имеющий единственную точку входа, которая позволяет обеспечить наличие в мониторе не более одного процесса в любой момент времени. Другие процессы, которые пытаются войти в монитор, присоединяются к очереди процессов, приостановленных в ожидании доступности монитора. После того как процесс вошел в монитор, он может временно приостановиться, выполнив вызов cwait (x); после этого процесс помещается в очередь процессов, ожидающих повторного входа в монитор при выполнении условия.

Если процесс, выполняющийся в мониторе, обнаруживает изменение переменной условия x, он выполняет операцию csignal(x), которая сообщает об обнаруженном изменении соответствующей очереди.

В качестве примера использования монитора вернемся к задаче производитель/потребитель с ограниченным буфером. В листинге 5.15 показано решение задачи с использованием монитора. Модуль монитора boundedbuffer управляет буфером,

использующимся для хранения и получения символов. Монитор включает две переменные условий: notfull истинно, если в буфере имеется место как минимум для одного символа, а notempty — если в буфере имеется по крайней мере один символ.

Мониторы с оповещением и широковещанием

Определение мониторов, данное Хоаром [HOAR74], требует, чтобы в случае, если очередь ожидания выполнения условия не пуста, при выполнении каким-либо процессом операции csignal для этого условия был немедленно запущен процесс, находящийся в указанной очереди. Таким образом, выполнивший операцию csignal процесс должен либо немедленно выйти из монитора, либо быть приостановленным.

У такого подхода имеется два недостатка.

Если выполнивший операцию csignal процесс не завершил свое пребывание в мониторе, то требуются два дополнительных переключения процессов: одно для приостановки данного процесса и второе для возобновления его работы, когда монитор станет доступен.

Планировщик процессов, связанный с сигналом, должен быть идеально надежен. При выполнении csignal процесс из соответствующей очереди должен быть немедленно активизирован, причем планировщик должен гарантировать, что до активизации никакой другой процесс не войдет в монитор (в противном случае условие, в соответствии с которым активизируется процесс, может успеть измениться). Так, например, в листинге 5.15, когда выполняется csignal (notempty), процесс из очереди nonempty должен быть активизирован до того, как новый потребитель войдет в монитор. Вот и другой пример: сбой процесса производителя может произойти непосредственно после того, как он добавит символ к пустому буферу, так что операция csignal не будет выполнена. В результате процессы в очереди notempty окажутся навечно заблокированными.

Лэмпсон (Lampson) и Ределл (Redell) разработали другое определение монитора для языка Mesa [LAMP80]. Их подход позволяет преодолевать описанные проблемы, а кроме того, предоставляет ряд полезных расширений концепции мониторов. Структура монитора Mesa использована и в языке программирования Modula-3 [NELS91]. В языке программирования Mesa примитив csignal заменен примитивом cnotify, который интерпретируется следующим образом. Когда процесс, выполняющийся в мониторе, вызывает cnotify(x), об этом оповещается очередь условия x, но выполнение вызвавшего cnotify процесса продолжается. Результат оповещения состоит в том, что процесс в начале очереди условия возобновит свою работу в ближайшем будущем, когда монитор окажется свободным. Однако поскольку нет гарантии, что некий другой процесс не войдет в монитор до упомянутого ожидающего процесса, при возобновлении работы наш процесс должен еще раз проверить, выполнено ли условие.

Инструкции if заменены циклами while; таким образом, будет выполняться как минимум одно лишнее вычисление переменной условия. Однако в этом случае отсутствуют лишние переключения процессов и не имеется ограничений на момент запуска ожидающего процесса после вызова cnotify.

Одной из полезных особенностей такого рода мониторов может быть связанное с каждым примитивом условия cnotify предельное время ожидания. Процесс, который прождал уведомления в течение предельного времени, помещается в список активных независимо от того, было уведомление о выполнении условия или нет. При активизации процесс проверяет, выполнено ли условие, и если да, то продолжает свою работу. Такая возможность предотвращает бесконечное голодание процесса в случае, когда другие процессы сбоят перед уведомлением о выполнении условия.

При использовании правила, согласно которому происходит уведомление процесса, а не его насильственная активизация, в систему команд можно включить примитив cbroadcast, который вызывает активизацию всех ожидающих процессов. Это может быть удобно в ситуациях, когда процесс не осведомлен о количестве ожидающих процессов. Предположим, например, что в программе производитель/потребитель функции append и

take могут работать с символьными блоками переменной длины. В этом случае, когда производитель добавляет в буфер блок символов, он не обязан знать, сколько символов готов потребить каждый из ожидающих процессов. Он просто выполняет инструкцию cbroadcast, и все ожидающие процессы получают уведомление о том, что они могут попытаться получить свою долю символов из буфера.

Кроме того, широковещательное сообщение может использоваться в том случае, когда процесс не в состоянии точно определить, какой именно процесс из ожидающих должен быть активизирован. Хорошим примером такой ситуации может служить диспетчер памяти. Допустим, у нас имеется байт свободной памяти, и некоторый процесс освобождает дополнительно h байт. Диспетчеру не известно, какой именно из ожидающих процессов сможет работать с k+j байт свободной памяти; следовательно, он должен использовать вызов cbroadcast, и все ожидающие процессы сами проверят, достаточно ли им освободившейся памяти.

Преимуществом монитора Лэмпсона-Редделла по сравнению с монитором Хоара является его меньшая подверженность ошибкам. При подходе Лэмпсона-Редделла, поскольку каждая процедура после получения сигнала проверяет переменную монитора с использованием цикла while, процесс может послать неверное уведомление или широковещательное сообщение, и это не приведет ошибке в программе, получившей сигнал (попросту убедившись, что ее зря активизировали, программа вновь перейдет в состояние ожидания).

Другим достоинством монитора Лэмпсона-Редделла является то, что он способствует использованию модульного подхода при создании программ. Изменение условий при использовании этого типа монитора не требует изменения всей системы сигналов, так как каждый процесс сам проверяет выполнение соответствующих условий при активизации.

2) Механизмы параллельных вычислений в Windows.

UNIX предоставляет различные механизмы для синхронизации и связи между процессами В этом разделе мы рассмотрим важнейшие из них:

- каналы;
- сообщения;
- разделяемую память;
- семафоры;
- сигналы.

Каналы, сообщения и разделяемая память обеспечивают обмен данными между процессами, в то время как семафоры и сигналы используются для инициации некоторых действий другого процесса.

Каналы

Каналы (pipes) являются одним из наиболее значительных вкладов UNIX в развитие операционных систем. Разработанные на основе концепции сопрограмм [RITC84], каналы представляют собой циклические буфера, которые позволяют двум процессам связываться друг с другом в соответствии с моделью производитель/потребитель. Следовательно, канал — не что иное, как очередь, работающая по принципу "первым вошел — первым вышел", запись в которую осуществляется одним процессом, а чтение — другим.

При создании канала он получает буфер определенного размера. При записи в канал при наличии свободного места соответствующий запрос удовлетворяется немедленно; в противном случае процесс блокируется. Аналогично блокируется процесс, пытающийся прочесть из канала большее количество информации, чем имеющееся в нем; в противном случае запрос на чтение выполняется немедленно. Операционная система обеспечивает взаимоисключения — одновременно доступ к каналу имеет только один процесс.

Существует два типа каналов: именованные и неименованные. Совместно использовать неименованные каналы могут только связанные друг с другом про-цессы; не связанные друг с другом процессы могут совместно использовать только именованные каналы.

Сообщения

Сообщение представляет собой блок текста определенного типа. UNIX для работы с системой передачи сообщений предоставляет процессам системные вызовы msgsnd и msgrcv. С каждым процессом связана очередь сообщений, функционирующая подобно почтовому ящику.

Указанный отправителем тип сообщения может быть использован получа¬телем как критерий отбора сообщений. Он может получать сообщения либо в соответствии с принципом "первым вошел — первым вышел", либо в соответствии с их типом. При попытке отправить сообщение в заполненную очередь выполнение процесса приостанавливается, так же как и при попытке прочесть сообщение из пустой очереди. Если же процесс пытается прочесть сообщение определенного типа, но такого сообщения в очереди нет, процесс не приостанавливается.

Разделяемая память

Наиболее быстрым видом связи между процессами, обеспечиваемым операционной системой UNIX, является разделяемая память. Это общий блок виртуальной памяти, совместно используемый многими процессами. Процессы читают информацию и записывают ее в разделяемую память с помощью тех же инструкций чтения и записи, что и при работе с другими частями своего виртуального пространства памяти. Права доступа (чтение и запись или только чтение) к разделяемой памяти предоставляются каждому из процессов в отдельности. Взаимоисключения не являются частью механизма разделяемой памяти и должны обеспечиваться процессами, использующими разделяемую память.

Семафоры

Система вызовов семафоров в UNIX System V представляет собой обобщение примитивов wait и signal, определенных в главе 5, "Параллельные вычисления: взаимоисключения и многозадачность". Все требуемые при работе с семафорами операции выполняются ядром автоматически; ни один процесс не может получить доступ к семафору, пока с ним выполняется операция, вызванная другим процессом.

Семафор состоит из следующих элементов.

Текущее значение семафора.

Идентификатор последнего процесса, работавшего с семафором.

Количество процессов, ожидающих, пока значение семафора не превысит текущее.

Количество процессов, ожидающих, пока значение семафора не станет равным нулю.

С семафором связаны очереди приостановленных процессов.

При создании семафоры принадлежат множествам. Множество может содержать как один так и несколько создаваемых семафоров. Системный вызов semctl позволяет установить значения всех семафоров множества одновременно. Кроме того, имеется системный вызов semop, в качестве аргумента которому передается список операций с семафорами (по одной для каждого семафора из множества). При этом вызове ядро выполняет указанные операции одновременно. Каждая из операций определяется значением sem_op.

Если sem_op положительно, ядро увеличивает значение семафора и активизирует все процессы, ожидающие увеличения значения семафора.

Если sem_op равно 0, ядро проверяет значение семафора. Если оно нулевое, то ядро переходит к выполнению операций со следующим семафором; в противном случае количество процессов, ожидающих обнуления семафора, увеличивается, и процесс приостанавливается до тех пор, пока значение семафора не станет равным нулю.

Если sem_op отрицательно, а его абсолютное значение не превышает значение семафора, ядро добавляет sem_op (отрицательное число!) к значению семафора. Если полученный результат равен нулю, ядро активизирует все процессы, ожидавшие обнуления значения семафора.

Если sem_op отрицательно, а его абсолютное значение больше значения семафора, ядро приостанавливает процесс до тех пор, пока значение семафора не увеличится.

Такое обобщение семафоров обеспечивает значительную гибкость при выполнении синхронизации и координации процессов.

Сигналы

Сигнал представляет собой программный механизм, информирующий процесс о наступлении асинхронного события. Сигнал подобен аппаратному прерыванию, но не использует систему приоритетов, т.е. все сигналы обрабатываются одинаково. Процесс получает сигналы по одному, без специального упорядочения.

Процессы могут посылать сигналы друг другу; в обмене сигналами может принимать участие и ядро. Доставка сигнала выполняется путем обновления поля в таблице процесса, которому послан данный сигнал. Поскольку каждый сигнал соответствует отдельному биту, сигналы одного типа не могут накапливаться в виде очереди. Сигнал обрабатывается сразу же после активизации процесса или возврата его из системного вызова. Процесс может ответить на сигнал выполнением некоторых действий по умолчанию (например, завершением работы), выполнить функцию обработки сигнала или проигнорировать его.

Параллельные вычисления в Windows

Windows 2000 (W2K) обеспечивает синхронизацию потоков как часть объектной архитектуры. Механизм, использованный W2K для реализации синхронизации, представляет собой семейство следующих объектов синхронизации.

- Процесс.
- Поток.
- Файл.
- Консольный ввод.
- Уведомление об изменении файлов.
- Мьютекс.
- Семафор.
- Событие.
- Таймер ожидания.

Первые четыре объекта в этом списке могут использоваться для целей синхронизации, хотя это и не основное их предназначение. Остальные типы объектов разработаны специально для поддержки синхронизации.

Экземпляр каждого из перечисленных объектов может находиться в одном из двух состояний — сигнальном (signaled) и несигнальном (unsignaled); поток освобождается при входе объекта в сигнальное состояние. Механизм достаточно прост: поток выполняет запрос на ожидание к исполнительной системе W2K с использованием дескриптора объекта синхронизации. Когда объект входит в сигнальное состояние, исполнительная система W2K освобождает все потоки, находящиеся в состоянии ожидания этого объекта.

2.5.3 Результаты и выводы:

Контрольные вопросы:

1) Что такое семафор, в чем его функция?
2) Какие 3 операции могут быть произведены над семафором?
3) Что такое очередь?
4) Что такое монитор, его функции?
5) Характеристики монитора

6) Какие механизмы для синхронизации и связи между процессами вы знаете?
7) Что такое каналы?
8) Работа разделяемой памяти
9) Что такое сигнал?

2.6 Практическое занятие №11-12 (4 часа).
Тема: «Устройства хранения данных. Структура основной памяти»

2.6.1 Задание для работы:

1) Флэш-носители.
2) Дисковая память.

2.6.2 Краткое описание проводимого занятия:

1) Перемещение. Защита. Совместное использование.

Перемещение

В многозадачной системе доступная основная память разделяется множеством процессов. Обычно программист не знает заранее, какие программы будут резидентно находиться в основной памяти во время работы разрабатываемой им программы. Кроме того, для максимизации загрузки процессора желательно иметь большой пул процессов, готовых к исполнению, для чего требуется возможность загрузки и выгрузки активных процессов из основной памяти. Требование, чтобы выгруженная из памяти программа была вновь загружена в то же самое место, где находилась и ранее, было бы слишком сильным ограничением. Крайне желательно, чтобы программа могла быть перемещена (relocate) в другую область памяти.

Таким образом, заранее неизвестно, где именно будет размещена программа, а кроме того, программа может быть перемещена из одной области памяти в другую при свопинге. Эти обстоятельства обуславливают наличие определенных технических требований к адресации, проиллюстрированных на рис. 7.1. На рисунке представлен образ процесса. Для простоты предположим, что образ процесса занимает одну непрерывную область основной памяти. Очевидно, что операционной системе необходимо знать местоположение управляющей информации процесса истека исполнения, а также точки входа для начала выполнения процесса. Поскольку управлением памятью занимается операционная система и она же размещает процесс в основной памяти, соответствующие адреса она получает автоматически. Однако помимо получения операционной системой указанной информации, процесс должен иметь возможность обращаться к памяти в самой программе. Так, команды ветвления содержат адреса, указывающие на команды» которые должны быть выполнены после них; команды обращения к данным — адреса байтов или слов, с которыми они работают. Так или иначе, но процессор и программное обеспечение операционной системы должны быть способны перевести ссылки в коде программы в реальные физические адреса, соответствующие текущему расположению программы в основной памяти.

Защита

Каждый процесс должен быть защищен от нежелательного воздействия других процессов, случайного или преднамеренного. Следовательно, код других процессов не должен иметь возможности без разрешения обращаться к памяти данного процесса для чтения или записи. Однако удовлетворение требованию перемещаемости усложняет задачу защиты. Поскольку расположение программы в основной памяти непредсказуемо, проверка абсолютных адресов во время компиляции невозможна. Кроме того, в большинстве языков программирования возможно динамическое вычисление адресов во время исполнения (например, вычисление адреса элемента массива или указателя на поле структуры данных). Следовательно, во время работы программы необходимо выполнять проверку всех обращений к памяти, генерируемых процессом, чтобы удостовериться, что

все они -только к памяти, выделенной данному процессу. К счастью, как вы увидите позже, механизмы поддержки перемещений обеспечивают и поддержку защиты.

Обычно пользовательский процесс не может получить доступ ни к какой части операционной системы — ни к коду, ни к данным. Код одного процесса не может выполнить команду ветвления, целевой код которой находится в другом процессе. Если не приняты специальные меры, код одного процесса не может получить доступ к данным другого процесса. Процессор должен быть способен прервать выполнение таких команд.

Заметим, что требования защиты памяти должны быть удовлетворены на уровне процессора (аппаратного обеспечения), а не на уровне операционной системы (программного обеспечения), поскольку операционная система не в состоянии предвидеть все обращения к памяти, которые будут выполнены программой. Даже если бы такое было возможно, сканирование каждой программы в поиске предлагаемых нарушений защиты было бы слишком расточительно с точки зрения использования процессорного времени. Следовательно, соответствующие возможности аппаратного обеспечения - единственный способ определения допустимости обращения к памяти (данным или коду) во время работы программы.

Совместное использование

Любой механизм защиты должен иметь достаточную гибкость, для того чтобы обеспечить возможность нескольким процессам обращаться к одной и той же области основной памяти. Например, если несколько процессов выполняют один и тот же машинный код, то будет выгодно позволить каждому процессу работать с одной и той же копией этого кода, а не создавать свою собственную. Процессам, сотрудничающим в работе над некоторой задачей, может потребоваться совместный доступ к одним и тем же структурам данных. Система управления памятью должна, таким образом, обеспечивать управляемый доступ к разделяемым областям памяти, при этом никоим образом не ослабляя защиту памяти. Как мы увидим позже, механизмы поддержки перемещений обеспечивают и поддержку совместного использования памяти.

2) Распределение памяти. Страничная организация. Сегментация.

Распределение памяти

Основной памяти для ее выполнения процессором. Практически во всех современных многозадачных системах эта задача предполагает использование сложной схемы, известной как виртуальная память. Виртуальная память, в свою очередь, основана на использовании одной или обеих базовых технологий — сегментов и страниц. Перед тем как перейти к рассмотрению этих методов организации виртуальной памяти, мы должны сперва познакомиться с более простыми методами (табл. 7.1)- Одна из приведенных в таблице технологий - распределение памяти - использовалась в различных вариациях в некоторых уже подзабытых к настоящему времени операционных системах. Две другие технологии — простая страничная организация и сегментация - сами по себе не используются, однако их рассмотрение в отрыве от виртуальной памяти упростит дальнейшее понимание предлагаемого материала.

Фиксированное распределение

В большинстве схем управления памятью мы будем полагать, что операционная система занимает некоторую фиксированную часть основной памяти и что остальная часть основной памяти доступна для использования многочисленным процессам. Простейшая схема управления этой доступной памятью — ее распределение на области с фиксированными границами.

Страничная организация.

Как разделы с разными фиксированными размерами, так и разделы переменного размера недостаточно эффективно используют память. Результатом работы первых становится внутренняя фрагментация, результатом работы последних— внешняя. Предположим, однако, что основная память разделена на одинаковые блоки относительно

небольшого фиксированного размера. Тогда блоки процесса, известные как страницы (pages), могут быть связаны со свободными блоками памяти, известными как кадры (frames), или фреймы. Каждый кадр может содержать одну страницу данных. При такой организации памяти, как вы узнаете из этого раздела, внешняя фрагментация отсутствует вовсе, а потери из-за внутренней фрагментации ограничены частью последней страницы процесса.

В любой момент времени некоторые из кадров памяти используются, а некоторые свободны. Операционная система поддерживает список свободных кадров. Процесс А, хранящийся на диске, состоит из четырех страниц. Когда приходит время загрузить этот процесс в память, операционная система находит четыре свободных кадра и загружает страницы процесса А в эти кадры (рис. 7.9,б). Затем загружаются процесс В, состоящий из трех страниц, и процесс С, состоящий из четырех страниц. После этого процесс В приостанавливается и выгружается из основной памяти. Позже наступает момент, когда все процессы в памяти оказываются заблокированы, и операционная система загружает в память новый процесс D, состоящий из пяти страниц.

Теперь предположим, что, как в только что рассмотренном выше примере, не имеется одной непрерывной области кадров, достаточной для размещения процесса целиком. Помешает ли это операционной системе загрузить процесс D? Нет, поскольку в этой ситуации можно воспользоваться концепцией логических адресов. Однако одного регистра базового адреса в этой ситуации недостаточно, и для каждого процесса операционная система должна поддерживать таблицу страниц. Таблица страниц указывает расположение кадров каждой страницы процесса. Внутри программы логический адрес состоит из номера страницы и смещения внутри нее. Вспомним, что в случае простого распределения логический адрес представляет собой расположение слова относительно начала программы, которое процессор транслирует в физический адрес. При страничной организации преобразование логических адресов в физические также остается задачей аппаратного уровня, решаемой процессором процессором. Теперь процессор должен иметь информацию о том, где находится таблица страниц текущего процесса. Представленный логический адрес (номер страницы и смещение) процессор превращает с использованием таблицы страниц в физический адрес (номер кадра, смещение).

Таким образом, описанная здесь простая страничная организация подобна фиксированному распределению. Отличия заключаются в достаточно малом размере разделов, которые к тому же могут не быть смежными.

Для удобства работы с такой схемой добавим правило, в соответствии с которым размер страницы (а, следовательно, и размер кадра) должен представлять собой степень 2. При использовании такого размера страниц легко показать, что относительный адрес, который определяется относительно начала программы, и логический адрес, представляющий собой номер кадра и смещение, идентичны. Соответствующий пример приведен на рис, 7,11. Здесь используется 16-битовый адрес и страницы размером 1 Кбайт = 1024 байт. Относительный адрес 1502 в бинарном виде записывается как 0000010111011110. При размере страницы в 1 Кбайт поле смещения требует 10 бит, оставляя 6 бит для номера страницы. Таким образом, программа может состоять максимум из 26 = 64 страниц по 1 Кбайт каждая. Как показано на рис. 7.11, относительный адрес 1502 соответствует смещению 478 (0111011110) на странице 1 (000001), что дает то же бинарное число 0000010111011110.

Использование страниц с размером, равным степени двойки, приводит к таким следствиям. Во-первых, схема логической адресации прозрачна для программиста, ассемблера и компоновщика. Каждый логический адрес (номер страницы и смещение) программы идентичен относительному адресу. Во-вторых, при этом относительно просто реализуется аппаратная функция преобразования адресов во время работы. Рассмотрим адрес из п+т бит, где крайние слева п бит представляют собой номер страницы, а крайние

справа т. бит — смещение. В нашем примере (рис. 7.11,б) n= 6 и m=10. Для преобразования адреса необходимо выполнить следующие шаги.

Выделить номер страницы, который представлен л левыми битами логического адреса.

Используя номер страницы в качестве индекса в таблице страниц процесса, найти номер кадра k.

Начальный физический адрес кадра — k x 2m , и интересующий нас физический адрес представляет собой это число плюс смещение. Такой адрес не надо вычислять — он получается в результате простого добавления номера кадра к смещению.

В нашем примере имеется логический адрес 0000010111011110, представляющий страницу номер 1 и смещение 478. Предположим, что эта страница размещена в кадре основной памяти номер 6 (бинарное представление — 000110). В таком случае физический адрес представляет собой кадр б, смещение 478, т.е. 0001100111011110 (рис. 7.12,а).

Итак, в случае простой страничной организации основная память разделяется на множество небольших кадров одинакового размера. Каждый процесс разделяется на страницы того же размера, что и кадры; малые процессы требуют меньшего количества кадров, большие — большего. При загрузке процесса в память все его страницы загружаются в свободные кадры, и информация о размещении страниц заносится в соответствующую таблицу. Такой подход позволяет избежать множества присущих распределению памяти проблем.

Сегментация

Альтернативным способом распределения пользовательской программы является сегментация. В этом случае программа и связанные с нею данные разделяются на ряд сегментов. Хотя и существует максимальный размер сегмента, на них не накладывается условие равенства размеров. Как и при страничной организации, логический адрес состоит из двух частей, в данном случае — номера сегмента и смещения.

Использованием сегментов разного размера этот способ похож на динамическое распределение памяти. Если не используются оверлеи и виртуальная память, то для выполнения программы все ее сегменты должны быть загружены в память; однако в отличие от динамического распределения в этом случае сегменты могут занимать несколько разделов, которые, к тому же, могут не быть смежными. Сегментация устраняет внутреннюю фрагментацию, однако, как и динамическое распределение, страдает от фрагментации внешней. Тем не менее ее степень снижается, в силу того что процесс разбивается на ряд небольших частей.

В то время как страничная организация невидима для программиста, сегментация, как правило, видима и обычно используется при размещении кода и данных в разных сегментах. При использовании принципов модульного программирования как код, так и данные могут быть дополнительно разбиты на сегменты. Главным недостатком при работе с сегментами является необходимость заботиться о том, чтобы размер сегмента не превысил максимальный.

Еще одно следствие того, что сегменты имеют разные размеры, состоит в отсутствии простого соотношения между логическими и физическими адресами. Аналогично страничной организации, схема простой сегментации использует таблицу сегментов для каждого процесса и список свободных блоков основной памяти. Каждая запись таблицы сегментов должна содержать стартовый адрес сегмента в основной памяти и его длину, чтобы обезопасить систему от использования некорректных адресов. При работе процесса адрес его таблицы сегментов заносится в специальный регистр, используемый аппаратным обеспечением. Рассмотрим адрес из n+т бит, где крайние слева n бит являются номером сегмента, а правые т бит — смещением. В нашем примере, помещенном на рис* 7.11,в, n = 4 и т = 12. Таким образом, максимальный размер сегмента

составляет 212 = 4096. Для трансляции адреса необходимо выполнение следующих действий.

Выделить из логического адреса п крайних слева битов, получив таким об¬разом номер сегмента.

Используя номер сегмента в качестве индекса в таблице сегментов процесса, найти физический адрес начала сегмента.

Сравнить смещение, представляющее собой крайние справа т бит, с длиной сегмента. Если смещение больше длины, адрес некорректен.

Требуемый физический адрес представляет собой сумму физического адреса начала сегмента и смещения.

В нашем примере имеется логический адрес 0001001011110000, представляющий собой сегмент номер 1, смещение 752. Предположим, что этот сегмент располагается в основной памяти начиная с физического адреса 0010000000100000. Тогда интересующий нас физический адрес равен 0010000000100000+001011110000 = 0010001100010000 (см. рис. 7.12,б).

Итак, в случае простой сегментации процесс разделяется на ряд сегментов, размер которых может быть разным. При загрузке процесса все его сегменты размещаются в свободных областях памяти, и соответствующая информация вносится в таблицу сегментов.

2.6.3 Результаты и выводы:

Контрольные вопросы:

1) Для чего нужно перемещение?
2) Для чего нужно защищать процесс от других процессов?
3) Принцип работы совместного пользования
4) Что такое фиксированное распределение?
5) Принципы работы страничной организации
6) Что такое сегментация?
7) В чем сходство между сегментацией и динамеческим распределением памяти?
8) Что такое физический адрес сегмента?

2.7 Практическое занятие №13-14 (4 часа).

Тема: «Устройства хранения данных. Структура основной памяти»

2.7.1 Задание для работы:

1) Интерфейсы COM
2) Интерфейсы USB

2.7.2 Краткое описание проводимого занятия:

1) Стратегия выборки. Стратегия размещения. Стратегия замещения.

Стратегия выборки

Стратегия выборки определяет, когда страница должна быть передана в основную память. Два основных варианта — по требованию и предварительно. При выборке по требованию страница передается в основную память только тогда, когда выполняется обращение к ячейке памяти, расположенной на этой странице. Если все прочие элементы системы управления памятью работают хорошо, то должно произойти следующее. Когда процесс только запускается, возникает поток прерываний обращений к странице, но далее срабатывает принцип локализации, и все большее количество обращений выполняется к недавно загруженным страницам. Соответственно, количество прерываний из-за отсутствия страницы снижается до весьма низкого уровня.

В случае предварительной выборки загружается не только страница, вызвавшая прерывание обращения. Предварительная выборка использует характеристики

большинства устройств вторичной памяти, таких, как диски, у которых имеется время поиска и задержка, связанная с вращением диска. Если страницы процесса расположены во вторичной памяти последовательно, то гораздо более эффективной будет загрузка в основную память нескольких последовательных страниц за один раз, чем загрузка этих же страниц по одной в течение некоторого промежутка времени. Естественно, эта стратегия не дает никакого выигрыша, если обращения к дополнительно загруженным страницам не происходит.

Предварительная выборка может применяться либо при первом запуске процесса к страницам, тем или иным способом, указываемым программистом, либо каждый раз при каждом прерывании обращения к странице. Последний случай кажется более предпочтительным, поскольку он прозрачен для программиста. Тем не менее, выгодность использования предварительной выборки не доказана [MAEK87].

Не следует путать предварительную выборку и свопинг. При выгрузке процесса из памяти и переводе его в приостановленное состояние из основной памяти удаляются все его резидентные страницы. При возобновлении выполнения процесса все его страницы, которые ранее находились в основной памяти, вновь возвращаются в нее.

Стратегия размещения

Стратегия размещения определяет, где именно в физической памяти будут располагаться части процесса. В случае "чистой" сегментации стратегия размещения является весьма важным вопросом, решения которого в виде стратегий первого подходящего, очередного подходящего и других рассматривались в главе 7 "Управление памятью". Однако для систем, использующих только страничную организацию или страничную организацию в сочетании с сегментацией, стратегия размещения обычно не так важна, поскольку аппаратная трансляция адреса и аппаратное обращение к памяти одинаково результативны при любых сочетаниях страница-кадр.

В так называемых многопроцессорных системах с неоднородным доступом к памяти (nonuniformmemoryaccess — NUMA) размещение является довольно важным вопросом, требующим всестороннего исследования. Обратиться к распределенной совместно используемой памяти может любой процессор, однако на время доступа к определенному физическому адресу влияет расстояние между процессором и модулем памяти. Таким образом, суммарная производительность в огромной степени зависит от того, насколько близко к процессору размещены обрабатываемые им данные [LAR092, BOL089, COX89]. В системах с неоднородным доступом к памяти, в соответствии со стратегией автоматического размещения страницы должны размещаться в модулях памяти, обеспечивающих наибольшую производительность.

Стратегия замещения

В большинстве публикаций, посвященных операционным системам, рассмотрение управления памятью включает в себя раздел, озаглавленный "Стратегия замещения", в котором говорится о выборе страниц в основной памяти для замещения их загружаемыми из вторичной памяти страницами. Эта тема достаточно сложна методологически, поскольку включает ряд взаимосвязанных вопросов.

• Какое количество кадров должно быть выделено каждому активному процессу.

• Должно ли множество страниц, которые потенциально могут быть замещены загружаемыми страницами, ограничиваться одним процессом или в качестве кандидатов на замещение могут рассматриваться все кадры страниц основной памяти.

• Какие именно страницы из рассматриваемого множества следует выбрать для замещения.

Первые два вопроса — из области управления резидентным множеством, о чем мы поговорим в следующем подразделе; термин же "стратегия замещения" мы будем использовать для обозначения третьего вопроса.

Вопросы стратегии замещения представляют собой, пожалуй, наиболее полно изученный за последние 20 лет аспект управления памятью- Когда все кадры основной

памяти заняты и нам требуется разместить новую страницу в процессе обработки прерывания из-за отсутствия страницы, стратегия замещения определяет, какая из находящихся в настоящее время в основной памяти страниц должна быть выгружена, чтобы освободить кадр для загружаемой страницы. Все стратегии направлены на то, чтобы выгрузить страницу, обращений к которой в ближайшем будущем не последует. В соответствии с принципом локализации часто наблюдается сильная корреляция между множеством страниц, к которым в последнее время были обращения, и множеством страниц, к которым будут обращения в ближайшее время. Таким образом, большинство стратегий пытаются определить будущее поведение программы на основеее прошлого поведения. При рассмотрении разных стратегий следует учитывать, что чем, более совершенный и интеллектуальный алгоритм использует стратегия, тем выше будут накладные расходы при его реализации.

2) Управление загрузкой. Стратегия очистки.Распределение памяти ядра.

Как отмечалось ранее, размер основной памяти со временем становится все больше, как, впрочем, и размер приложений. Утешением может служить то, что размеры кэшей также увеличиваются. Большие — до нескольких мегабайтов — кэши в настоящее время вполне возможны [BORG90]. При использовании кэшей большого размера замещение страниц виртуальной памяти может влиять на производительность. Если кадр страницы, выбранный для замещения, располагается в кэше, то вместе с потерей страницы из блока кэша теряется весь блок.

В системах с использованием буферизации того или иного вида производительность кэша можно увеличить путем добавления к стратегии замещения стратегию размещения страниц в буфере. Большинство операционных систем размещают страницы в буфере в произвольных кадрах, как правило, с использованием алгоритма "первым вошел — первым вышел". Исследования в [KESS92] показали, что правильный выбор стратегии размещения может привести к уменьшению неуспешных поисков в кэше на 10-20%.

В [KESS92] описаны исследования ряда алгоритмов размещения в зависимости от структуры кэша и используемых стратегий. Суть этих стратегий состоит в размещении последовательных страниц в основной памяти таким образом, чтобы минимизировать количество кадров страниц, отображаемых в одни и те же слоты кэша.

Стратегия очистки

Стратегия очистки является противоположностью стратегии выборки. Ее задача состоит в определении момента, когда измененная страница должна быть записана во вторичную память. Два основных ее метода — очистка по требованию и предварительная очистка. При очистке по требованию страница записывается во вторичную память только тогда, когда она выбирается для замещения. Предварительная очистка записывает модифицированные страницы до того, как потребуются занимаемые ими кадры, так что эти страницы могут записываться пакетами.

Имеется опасность прямолинейного следования любой из стратегий. При предварительной очистке записанная страница остается в основной памяти до тех пор, пока ее не удалит оттуда алгоритм замещения. Предварительная очистка позволяет записывать страницы пакетами, но не имеет смысла записывать сотни или тысячи страниц только для того, чтобы убедиться, что до замещения они вновь успели модифицироваться. Пропускная способность вторичной памяти ограничена и не должна засоряться излишними операциями очистки.

С другой стороны, при очистке по требованию запись модифицированной страницы сопровождается чтением новой страницы, так что, несмотря на минимизацию записей страниц, прерывание обращения может вызывать пересылку двух страниц между основной и вторичной памятью и тем самым снижать эффективность использования процессора.

Улучшенный подход включает буферизацию страниц, что позволяет принять следующую стратегию: очищать только замещаемые страницы, но при этом разделить операции очистки и замещения. При использовании буферизации страниц замещаемые страницы могут находиться в двух списках: модифицированных и не модифицированных страниц. Страницы из списка модифицированных могут периодически записываться пакетами и переноситься в список не модифицированных. Страница из списка не модифицированных страниц может либо быть удалена из него при обращении к ней, либо потеряна при загрузке в ее кадр новой страницы.

Распределение памяти ядра

Ядро в процессе работы часто генерирует и уничтожает маленькие таблицы и буфера, память для каждого из которых выделяется динамически. В [VAHA96] перечислены следующие примеры.

• Преобразование имени пути может запросить буфер для копирования имени из пользовательского пространства.

• Подпрограмма allocb( ) выделяет буфера произвольного размера.

• Ряд реализации UNIX выделяют "зомби"-структуры для хранения информации о состоянии выхода и использовании ресурсов завершенными процессами.

• В SVR4 и Solaris ядро динамически распределяет множество объектов (таких, как структуры процессов, блоки дескрипторов файлов и др.).

Размер большинства этих блоков гораздо меньше типичного размера страницы памяти и, соответственно, использование страничного механизма в данном случае крайне неэффективно. В SVR4 используется модификация системы двойников (описанной в разделе 7.2).

Стоимость выделения свободного блока памяти в системе двойников меньше, чем в случае использования стратегий первого или наилучшего подходящего [KNUT97]. Однако при управлении памятью ядра выделение и освобождение памяти должно выполняться с максимально возможной скоростью. Недостатком же системы двойников являются затраты времени на разделение и слияние блоков.

Беркли (Barkley) и Ли (Lee) из AT&T предложили модификацию, известную как "ленивая" система двойников [BARK89], которая принята в SVR4. Авторами замечено, что UNIX часто демонстрирует устойчивое состояние памяти ядра, т.е. количество требующихся блоков определенного размера мало меняется со временем. Таким образом, вполне возможна ситуация, когда освобождающийся блок размером 2i сливается со своим двойником в блок размером 2i+1, который тут же вновь разделяется на два блока размером 2i в соответствии с запросом системы. Чтобы избежать излишних слияний и разделений блоков, слияние освобожденных блоков откладывается до того момента, когда оно оказывается действительно необходимым (и тогда производится максимально возможное количество слияний блоков).

В модифицированной таким образом системе двойников используются следующие параметры:

Ni — текущее количество блоков размером 2i;

Ai — текущее количество занятых блоков размером2i;

Gi — текущее количество глобально свободных блоков размером 2i (Это блоки, пригодные для слияния со своими двойниками. Когда двойник такого блока становится глобально свободным, эти два блока сливаются в глобально свободный блок размером 2i+1. Все свободные блоки в системе двойников могут рассматриваться как глобально свободные);

Li — текущее количество локально свободных блоков размером2i (Это блоки, не пригодные для слияния. Даже если двойник такого блока становится свободным, эти два блока не сливаются, а остаются в ожидании последующих запросов на блоки данного размера.).

Выполняется следующее соотношение:

Ni= Ai+ Gi+ Li

В целом данная система двойников пытается поддерживать пул локально свободных блоков и производит слияние, только когда количество локально свободных блоков превышает предопределенный порог (при наличии слишком большого количества локально свободных блоков возрастает вероятность недостатка блоков большего размера для удовлетворения требований системы). В основном при освобождении блока слияние не выполняется, что минимизирует накладные расходы. Никаких различий между локально и глобально свободными блоками при выделении блока в ответ на запрос системы не делается.

Для слияния используется критерий, согласно которому количество локально свободных блоков данного размера не должно превышать количество занятых блоков этого размера (т.е. должно выполняться условие Li<Ai). Это вполне разумный принцип для ограничения количества локально свободных блоков; эксперименты, описанные в [BARK89], подтверждают, что такая схема приводит к значительному снижению стоимости распределения памяти.

Для реализации описанной схемы ее авторы определили переменную задержки Di = 4 – Li = Ni - 2Li –Gi . Алгоритм схемы приведен в листинге 8.1.

2.7.3 Результаты и выводы:

Контрольные вопросы:

1) Объяснить принцип работы стратегии выборки
2) Отличие стратегии выборки от свопинга
3) Принцип работы стратегии размещения
4) Какие вопросы рассматриваются в стратегии замещения
5) Основные методы стратегии очистки
6) Распределение памяти ядра

2.8 Практическое занятие №15-16 (4 часа).

Тема: «Аудиосистема ПК. Коммуникационные устройства»

2.8.1 Задание для работы:

1) Сервера для рабочей группы.
2) Многопроцессорные комплексы.

2.8.2 Краткое описание проводимого занятия:

1) Использование приоритетов.

Во многих системах каждому процессу присвоен некоторый приоритет, и планировщик всегда должен среди процессов выбирать тот, у которого приоритет наибольший. На рис. 9.4 показано использование приоритетов. Для большей ясности диаграмма упрощена и игнорирует существование нескольких очередей заблокированных или приостановленных процессов (ср. с рис. 3.5,а). Вместо одной очереди готовых к исполнению процессов у нас имеется их множество, упорядоченное по убыванию приоритета: RQ0, RQ1, ..., RQn, т.е.

Приоритет[RQi]> Приоритет[RQj] при i<j.2

При выборе процесса планировщик начинает с очереди процессов с наивысшим приоритетом (RQ0). Если в очереди имеются один или несколько процессов, процесс для работы выбирается с использованием некоторой стратегии планирования. Если очередь RQ0 пуста, рассматривается очередь RQ1 и т.д.

Одна из основных проблем в такой чисто приоритетной схеме планирования состоит в том, что процессы с низким приоритетом могут оказаться в состоянии голодания. Это будет происходить при постоянном поступлении новых готовых к

выполнению процессов с высоким приоритетом. Если такое поведение нежелательно, приоритет процесса может снижаться при его выполнении (пример такой стратегии планирования будет приведен позже).

2) Справедливое планирование.

Все рассмотренные алгоритмы планирования рассматривают множество готовых к выполнению процессов как единый пул, из которого выбирается очередной процесс для выполнения. Этот пул может быть разделен по степени приоритета процессов, но в противном случае он остается гомогенным.

Однако в многопользовательских системах при организации приложений или заданий отдельных пользователей как множества процессов (или потоков) у них имеется структура, не распознаваемая традиционными планировщиками. С точки зрения пользователя, важно не то, как будет выполняться отдельный процесс, а то, как будет выполняться множество процессов, составляющих единое приложение. Таким образом, было бы неплохо, если бы планирование осуществлялось с учетом наличия таких множеств процессов. Данный подход в целом известен как справедливое (fair-share) планирование. Эта же концепция может быть распространена на группы пользователей, даже если каждый из пользователей представлен единственным процессом. Например, в системе с разделением времени мы можем рассматривать всех пользователей данного отдела как членов одной группы. Планировщик принимает решения с учетом необходимости предоставить каждой группе пользователей по возможности одинаковый сервис. Таким образом, если в системе находится много пользователей из одного отдела, то изменение времени отклика должно в первую очередь коснуться именно пользователей этого отдела, не затрагивая прочих пользователей.

Термин справедливое планирование указывает на философию, лежащую в основе такого планирования. Каждому пользователю назначен определенный вес, который определяет долю использования системных ресурсов данным пользователем. В частности, каждый пользователь использует процессор. Данная схема работает более или менее линейно, так что если вес пользователя A в два раза превышает вес пользователя B, то в течение достаточно длительного промежутка времени пользователь A должен выполнить в два раза большую работу, чем пользователь B. Цель справедливого планирования состоит в отслеживании использования ресурсов и предоставлении меньшего количества ресурсов тому пользователю, который уже получил лишнее, и большего количества — тому, чья доля оказалась меньше справедливой.

Был предложен ряд алгоритмов справедливого планирования [HERN84, KAY88, WOOD86]. В этом разделе мы рассмотрим описанную в [HENR84] схему планирования, реализованную в ряде систем UNIX. Эта схема известна как справедливый планировщик (fair-sharescheduler — FSS). FSS при принятии решения рассматривает историю выполнения связанной группы процессов вместе с индивидуальными историями выполнения каждого процесса. Система разделяет пользовательское сообщество на множество групп со справедливым планированием и распределяет процессорное время между ними. Так, если у нас имеется четыре группы, каждая из них получит по 25% процессорного времени. В результате каждая группа обеспечивается виртуальной системой, работающей, соответственно, медленнее, чем система в целом.

Планирование осуществляется исходя из приоритетов с учетом приоритета процесса, недавнего использования им процессора и недавнего использования процессора группой, к которой он принадлежит. Чем больше числовое значение приоритета, тем ниже сам приоритет. Для процесса j из группы k применимы следующие формулы:

Каждому процессу назначается базовый приоритет. Приоритет процесса снижается по мере использования им процессора, так же как и по мере использования процессора группой в целом. В случае использования процессора группой среднее значение

нормализуется делением на вес группы. Чем больший вес назначен группе, тем меньше использование ею процессора влияет на приоритет.

В табл. 9.7 приведен пример, в котором процесс A находится в одной группе, а процессы B и C — в другой; вес каждой группы равен 0.5. Предположим, что все процессы ориентированы на вычисления и всегда готовы к выполнению. Базовый приоритет всех процессов – 60. Степень использования процессора определяется следующим образом: процессор прерывается 60 раз в секунду; при каждом прерывании увеличивается значение счетчика использования процессора текущего процесса, так же как и соответствующего счетчика группы. Один раз в секунду происходит перерасчет приоритетов.

В приведенной схеме первым запускается процесс A. Позже его вытесняет другой. Процессы B и C теперь имеют более высокий по сравнению с A приоритет, и на выполнение передается процесс B. По окончании второй единицы времени наивысший приоритет снова имеет процесс A, который и передается на выполнение. После этого ситуация повторяется с тем отличием, что теперь наивысший приоритет имеет процесс C. Очередность выполнения процессов такова: A, B, A, C, A, B, A, C, ... В результате 50% времени процессор занят выполнением процесса A, и 50% — выполнением процессов B и C.

2.8.3 Результаты и выводы:

Ответить на контрольные вопросы:

1) Что такое приоритет процесса?
2) Какова задача планировщика?
3) В чем смысл термина "состояние голодания процесса"
4) Перечислить преимущества справедливого планирования

2.9 Практическое занятие №17-20 (8 часов).

Тема: «Принципы построения процессора. Структура машинных команд и способы адресации**»**

2.9.1 Задание для работы:

Оперативная память. Назначение, принцип работы, технические характеристики маркировка

2.9.2 Краткое описание проводимого занятия:

Практическое занятие проводится в формах диспута и опроса по вопросам занятия

3.10 Практическое занятие №21-24 (4 часа).

Тема: «Современные микропроцессоры. Порядок выполнения машинных команд**»**

2.10.1 Задание для работы:

ПЗУ BIOS

2.10.2 Краткое описание проводимого занятия:

Практическое занятие проводится в формах диспута и опроса по вопросам занятия

2.11 Практическое занятие №25-26 (4 часа).

Тема: «Организация системы прерываний. Организация перехода к прерывающей программе. Принципы организации ввода-вывода**»**

2.11.1 Задание для работы:

Изучение работы стековой памяти в процессоре 8086, изучение способов адресации в процессоре 8086 и формирования исполнительного адреса

2.11.2 Краткое описание проводимого занятия:
Практическое занятие проводится в формах диспута и опроса по вопросам занятия

2.12 Практическое занятие №27-29 (6 часов).
Тема: «Архитектура системной платы. Установка и конфигурирование компонентов**»**

2.12.1 Задание для работы:

Обработка прерываний в микропроцессоре 8086.

2.12.2 Краткое описание проводимого занятия:
Практическое занятие проводится в формах диспута и опроса по вопросам занятия

2.13 Практическое занятие №30-32 (6 часов).
Тема: «Шины расширения. Шина USB**»**

2.13.1 Задание для работы:

Язык Ассемблера. Основные команды.

2.13.2 Краткое описание проводимого занятия:
Практическое занятие проводится в формах диспута и опроса по вопросам занятия

2.14 Практическое занятие №33-35 (6 часов).
Тема: «Параллельный интерфейс. Последовательный интерфейс**»**

2.14.1 Задание для работы:

Файловая система. Понятие, виды, структура.

2.14.2 Краткое описание проводимого занятия:
Практическое занятие проводится в формах диспута и опроса по вопросам занятия