Билет №15

1)Компораторы

Компаратор (аналоговых сигналов) — электронная схема, принимающая на свои входы два аналоговых сигнала и выдающая логическую «1», если сигнал на прямом входе («+») больше чем на инверсном входе («−»), и логический «0», если сигнал на прямом входе меньше, чем на инверсном входе.

Простейший компаратор представляет собой дифференциальный усилитель. Компаратор отличается от линейного операционного усилителя (ОУ) устройством и входного, и выходного каскадов:

- Входной каскад компаратора должен выдерживать широкий диапазон входных напряжений между инвертирующим и неинвертирующим входами, вплоть до размаха питающих напряжений, и быстро восстанавливаться при изменении знака этого напряжения. В ОУ, охваченном обратной связью, это требование некритично, так как дифференциальное входное напряжение измеряется милливольтами и микровольтами.
- Выходной каскад компаратора выполняется совместимым по уровням и токам с конкретным типом логических схем (ТТЛ, ЭСЛ и т. п.). Возможны выходные каскады на одиночном транзисторе с открытым коллектором (совместимость с ТТЛ и КПОМ логикой).

При подаче эталонного напряжения на инвертирующий вход, входной сигнал подаётся на неинвертирующий вход и компаратор является неинвертирующим (повторителем, буфером).

2)Правило нуля и единицы

Если сеть компараторов с га входами правильно упорядочивает все 2п возможных последовательностей нулей и единиц, то она является сортирующей, то есть правильно упорядочивает любую числовую последовательность.

http://reslib.com/book/Algoritmi__postroenie_i_analiz/592#592 - битонический сортировщик
http://lib.mexmat.ru/books/11254/s7 - корман

Билет №25

ПРИНЦИПЫ И МЕТОДЫ ЗАЩИТЫ ИНФОРМАЦИИ

Создание систем информационной безопасности (СИБ) в информационных системах (ИС) основывается на следующих принципах: Системный подход к построению системы защиты, означающий оптимальное сочетание взаимосвязанных организационных, программных, аппаратных, физических и других свойств, подтвержденных практикой создания отечественных и зарубежных систем защиты и применяемых на всех этапах технологического цикла обработки информации. Принцип непрерывного развития системы. Этот принцип, являющийся одним из основополагающих для компьютерных информационных систем, еще более актуален для СИБ. Способы реализации угроз информации в ИТ непрерывно совершенствуются, а потому обеспечение безопасности ИС не может быть одноразовым актом. Это непрерывный процесс, заключающийся в обосновании и реализации наиболее рациональных методов, способов и путей

совершенствования СИБ, непрерывном контроле, выявлении ее узких и слабых мест, потенциальных каналов утечки информации и новых способов несанкционированного доступа.
Обеспечение надежности системы защиты, т. е. невозможность снижения уровня надежности при возникновении в системе сбоев, отказов, преднамеренных действий взломщика или непреднамеренных ошибок пользователей и обслуживающего персонала.
Обеспечение контроля за функционированием системы защиты, т.е. создание средств и методов контроля работоспособности механизмов защиты.
Обеспечение всевозможных средств борьбы с вредоносными программами.
Обеспечение экономической целесообразности использования системы. защиты, что выражается в превышении возможного ущерба ИС и ИТ от реализации угроз над стоимостью разработки и эксплуатации СИБ.
В результате решения проблем безопасности информации современные ИСИнтерполяция и аппроксимация кривых и поверхностей должны обладать следующими основными признаками:
- наличием информации различной степени конфиденциальности;
-обеспечением криптографической защиты информации различной степени конфиденциальности при передаче данных;
- обязательным управлением потоками информации, как в локальных сетях, так и при передаче по каналам связи на далекие расстояния;
- наличием механизма регистрации и учета попыток несанкционированного доступа, событий в ИС и документов, выводимых на печать;
- обязательным обеспечением целостности программного обеспечения и информации в ИТ;
- наличием средств восстановления системы защиты информации;
- наличием физической охраны средств вычислительной техники и магнитных носителей;
- наличием специальной службы информационной безопасности системы.
Вся совокупность технических средств подразделяется на аппаратные и физические. Аппаратные средства — устройства, встраиваемые непосредственно в вычислительную технику, или устройства, которые сопрягаются с ней по стандартному интерфейсу. Физические средства

включают различные инженерные устройства и сооружения, препятствующие физическому проникновению злоумышленников на объекты защиты и осуществляющие защиту персонала (личные средства безопасности), материальных средств и финансов, информации от противоправных действий. Примеры физических средств: замки на дверях, решетки на окнах, средства электронной охранной сигнализации и т.п.
Программные средства — это специальные программы и программные комплексы, предназначенные для защиты информации в ИС.
Из средств ПО системы защиты необходимо выделить еще программные средства, реализующие механизмы шифрования (криптографии).
Организационные средства осуществляют своим комплексом регламентацию производственной деятельности в ИС и взаимоотношений исполнителей на нормативно-правовой основе таким образом, что разглашение, утечка и несанкционированный доступ к конфиденциальной информации становится невозможным или существенно затрудняется за счет проведения организационных мероприятий.
Законодательные средства защиты определяются законодательными актами страны, которыми регламентируются правила пользования, обработки и передачи информации ограниченного доступа и устанавливаются меры ответственности за нарушение этих правил. Морально-этические средства защиты включают всевозможные нормы поведения, которые традиционно сложились ранее, складываются по мере распространения ИС в стране и в мире или специально разрабатываются. Морально-этические нормы могут быть неписаные (например, честность) либо оформленные в некий свод (устав) правил или предписаний. Эти нормы, как правило, не являются законодательно утвержденными, но поскольку их несоблюдение приводит к падению престижа организации, они считаются обязательными для исполнения.

Технические каналы утечки информации

Технический канал утечки информации представляет собой совокупность объекта технической разведки, физической среды распространения информативного сигнала и средств, которыми добывается защищаемая информация. Техническая защита конфиденциальной информации – защита информации некриптографическими методами, направленными на предотвращение утечки защищаемой информации по техническим каналам, от несанкционированного доступа к ней и от специальных воздействий на информацию в целях ее уничтожения, искажения или блокирования.
Источником информации (объектом разведки) являются технические средства обработки информации (ОТСС, ВТСС), средства вычислительной техники, линии связи и человеческая речь.

Любой вид перечисленной информации распространяется в какой - либо среде (воздушная среда, инженерные коммуникации, линии связи, ограждающие конструкции), при этом на среду распространения воздействуют различные помехи (например, излишняя зашумленность помещения, электромагнитные помехи и т.д.). В итоге конечным пунктом будет являться приемник сигнала (разведка ПЭМИН, акустическая речевая разведка, разведка электрических сигналов речевого диапазона частот). Выяснив данные составляющие, можно представить путь информативного сигнала от источника по среде распространения к приемнику данного сигнала.

К техническим каналам утечки информации относят акустический (непредномеренное прослушивание, виброакустический канал, электроакустический канал), канал утечки за счет побочных электромагнитных излучений, а так же использование различных закладных устройств и средств разведки.

Защита информации от утечки по техническим каналам

- специальные проверки и специальные исследования аппаратных технических средств, позволяющие обнаружить возможно внедренные в них электронные средства съема информации и определить опасные зоны возможного перехвата информации по каналу
- защита выделенных посещений, в которых ведутся секретные переговоры(инструментальные измерения уровней звукоизоляции, специальные исследования на электроаккустические преобразования,Интерполяция и аппроксимация кривых и поверхностей высокочастотное навязывание, проектирование, монтаж и настройка систем защиты речевой информации акустическому, виброакустическому и оптоэлектронному каналам)
- поставка, монтаж и накладка средств защиты информации от утечки по каналу, сетям электропитания и заземления
- защита цифровых телефонных аппаратов путем установки устройств блокирования несанкционированного включения микрофонов цифрового телефонного аппарата
- защита телефонных аппаратов по каналу электроакустических преобразований и высокочастотного навязывания

Результат: обеспечение защиты информации от утечки по техническим каналам, защиты речевой информации в выделенных помещениях.

Билет №26

Симметри́чные криптосисте́мы (также **симметричное шифрование**, **симметричные шифры**) — способ шифрования, в котором для шифрования и расшифровывания применяется один и тот же криптографический ключ. До изобретения схемы асимметричного шифрования единственным существовавшим способом являлось

симметричное шифрование. Ключ алгоритма должен сохраняться в секрете обеими сторонами. Алгоритм шифрования выбирается сторонами до начала обмена сообщениями.

Основные сведения

Алгоритмы шифрования и дешифрования данных широко применяются в компьютерной технике в системах сокрытия конфиденциальной и коммерческой информации от злонамеренного использования сторонними лицами. Главным принципом в них является условие, что *передатчик и приемник заранее знают алгоритм шифрования*, а также ключ к сообщению, без которых информация представляет собой всего лишь набор символов, не имеющих смысла.

Классическим примером таких алгоритмов являются **симметричные криптографические алгоритмы**, перечисленные ниже:

- Простая перестановка
- Одиночная перестановка по ключу
- Двойная перестановка
- Перестановка "Магический квадрат"

Простая перестановка

Простая перестановка без ключа — один из самых простых методов шифрования. Сообщение записывается в таблицу по столбцам. После того, как открытый текст записан колонками, для образования шифровки он считывается по строкам. Для использования этого шифра отправителю и получателю нужно договориться об общем ключе в виде размера таблицы. Объединение букв в группы не входит в ключ шифра и используется лишь для удобства записи не смыслового текста.

Одиночная перестановка по ключу

Более практический метод шифрования, называемый одиночной перестановкой по ключу очень похож на предыдущий. Он отличается лишь тем, что колонки таблицы переставляются по ключевому слову, фразе или набору чисел длиной в строку таблицы.

Двойная перестановка

Для дополнительной скрытности можно повторно шифровать сообщение, которое уже было зашифровано. Этот способ известен под названием двойная перестановка. Для этого размер второй таблицы подбирают так, чтобы длины ее строк и столбцов были другие, чем в первой таблице. Лучше всего, если они будут взаимно простыми. Кроме того, в первой таблице можно переставлять столбцы, а во второй строки. Наконец, можно заполнять таблицу зигзагом, змейкой, по спирали или каким-то другим способом. Такие способы заполнения таблицы если и не усиливают стойкость шифра, то делают процесс шифрования гораздо более занимательным.

Перестановка «Магический квадрат»

Магическими квадратами называются квадратные таблицы со вписанными в их клетки последовательными натуральными числами от 1, которые дают в сумме по каждому столбцу, каждой строке и каждой диагонали одно и то же число. Подобные квадраты широко применялись для вписывания шифруемого текста по приведенной в них нумерации. Если потом выписать содержимое таблицы по строкам, то получалась шифровка перестановкой букв. На первый взгляд кажется, будто магических квадратов очень мало. Тем не менее, их число очень быстро возрастает с увеличением размера квадрата. Так,

существует лишь один магический квадрат размером 3 x 3, если не принимать во внимание его повороты. Магических квадратов 4 x 4 насчитывается уже 880, а число магических квадратов размером 5 x 5 около 250000. Поэтому магические квадраты больших размеров могли быть хорошей основой для надежной системы шифрования того времени, потому что ручной перебор всех вариантов ключа для этого шифра был немыслим.

Основные требования к алгоритмам асимметричного шифрования

Создание *алгоритмов асимметричного шифрования* является величайшим и, возможно, единственным революционным достижением в истории криптографии.

Алгоритмы шифрования с *открытым ключом* разрабатывались для того, чтобы решить две наиболее трудные задачи, возникшие при использовании симметричного шифрования.

Первой задачей является распределение ключа. При симметричном шифровании требуется, чтобы обе стороны уже имели общий ключ, который каким-то образом должен быть им заранее передан. Диффи, один из основоположников шифрования с *открытым ключом*, заметил, что это требование отрицает всю суть криптографии, а именно возможность поддерживать всеобщую секретность при коммуникациях.

Второй задачей является необходимость создания таких механизмов, при использовании которых невозможно было бы подменить кого-либо из участников, т.е. нужна *цифровая подпись*. При использовании коммуникаций для решения широкого круга задач, например в коммерческих и частных целях, электронные сообщения и документы должны иметь эквивалент подписи, содержащейся в бумажных документах. Необходимо создать метод, при использовании которого все участники будут убеждены, что электронное сообщение было послано конкретным участником. Это более сильное требование, чем аутентификация

Какой алгоритм лучше — симметричный или асимметричный? Вопрос не вполне корректен, поскольку предусматривает использование одинаковых критериев при сравнении криптосистем с секретным и открытым ключами. А таких критериев просто не существует.

Тем не менее, дебаты относительно достоинств и недостатков двух основных видов криптосистем ведутся давно, начиная с момента изобретения первого алгоритма с открытым ключом. Отмечено, что симметричные криптографические алгоритмы имеют меньшую длину ключа и работают быстрее, чем асимметричные.

Однако, по мнению американского криптолога У. Диффи — одного из и изобретателей криптосистем с открытым ключом— их следует рассматривав не как совершенно новую разновидность универсальных криптосистем. Криптография с открытым ключом и криптография с секретным ключом предназначены для решения абсолютно разных проблем, связанных с засекречиванием информации. Симметричные криптографические алгоритмы служат для шифрования данных, они работают на несколько порядком быстрее, чем асимметричные алгоритмы. Однако криптография с открытым ключом успешно используется в таких областях, для которых криптография с секретным ключом подходит

плохо, — например, при работе с ключами и с подавляющим большинством криптографических протоколов.

Криптографический протокол (англ. *Cryptographic protocol*) — это абстрактный или конкретный протокол, включающий набор криптографических алгоритмов. В основе протокола лежит набор правил, регламентирующих использование криптографических преобразований и алгоритмов в информационных процессах.

- Протоколы шифрования / расшифрования
- Протоколы электронной цифровой подписи (ЭЦП)
- Протоколы идентификации / аутентификации
- Протоколы аутентифицированного распределения ключей

Протоколы шифрования / расшифрования В основе протокола этого класса содержится некоторый симметричный или асимметричный алгоритм шифрования/расшифрования. Алгоритм шифрования выполняется на передаче отправителем сообщения, в результате чего сообщение преобразуется из открытой формы в шифрованную. Алгоритм расшифрования выполняется на приёме получателем, в результате чего сообщение преобразуется из шифрованной формы в открытую. Так обеспечивается свойство конфиденциальности.

Для обеспечения свойства целостности передаваемых сообщений симметричные алгоритмы шифрования / расшифрования, обычно, совмещаются с алгоритмами вычисления имитозащитной вставки (ИЗВ) на передаче и проверки ИЗВ на приёме, для чего используется ключ шифрования. При использовании асимметричных алгоритмов шифрования / расшифрования свойство целостности обеспечивается отдельно путем вычисления электронной цифровой подписи (ЭЦП) на передаче и проверки ЭЦП на приёме, чем обеспечиваются также свойства безотказности и аутентичности принятого сообщения.

Протоколы электронной цифровой подписи (ЭЦП) В основе протокола этого класса содержится некоторый алгоритм вычисления ЭЦП на передаче с помощью секретного ключа отправителя и проверки ЭЦП на приёме с помощью соответствующего открытого ключа, извлекаемого из открытого справочника, но защищенного от модификаций. В случае положительного результата проверки протокол, обычно, завершается операцией архивирования принятого сообщения, его ЭЦП и соответствующего открытого ключа. Операция архивирования может не выполняться, если ЭЦП используется только для обеспечения свойств целостности и аутентичности принятого сообщения, но не безотказности. В этом случае, после проверки, ЭЦП может быть уничтожена сразу или по прошествии ограниченного промежутка времени ожидания.

Протоколы идентификации / аутентификации

В основе протокола идентификации содержится некоторый алгоритм проверки того факта, что идентифицируемый объект (пользователь, устройство, процесс, …), предъявивший некоторое имя (идентификатор), знает секретную информацию, известную только заявленному объекту, причем метод проверки является, конечно, косвенным, то есть без предъявления этой секретной информации.

Обычно с каждым именем (идентификатором) объекта связывается перечень его прав и полномочий в системе, записанный в защищенной базе данных. В

этом случае протокол идентификации может быть расширен до протокола аутентификации, в котором идентифицированный объект проверяется на правомочность заказываемой услуги.
Если в протоколе идентификации используется ЭЦП, то роль секретной информации играет секретный ключ ЭЦП, а проверка ЭЦП осуществляется с помощью открытого ключа ЭЦП, знание которого не позволяет определить соответствующий секретный ключ, но позволяет убедиться в том, что он известен автору ЭЦП.

Протоколы аутентифицированного распределения ключей

Протоколы этого класса совмещают аутентификацию пользователей с протоколом генерации и распределения ключей по каналу связи. Протокол имеет двух или трёх участников; третьим участником является центр генерации и распределения ключей (ЦГРК), называемый для краткости сервером S. Протокол состоит из трёх этапов, имеющих названия: генерация, регистрация и коммуникация. На этапе генерации сервер S генерирует числовые значения параметров системы, в том числе, свой секретный и открытый ключ. На этапе регистрации сервер S идентифицирует пользователей по документам (при личной явке или через уполномоченных лиц), для каждого объекта генерирует ключевую и/или идентификационную информацию и формирует маркер безопасности, содержащий необходимые системные константы и открытый ключ сервера S (при необходимости). На этапе коммуникации реализуется собственно протокол аутентифицированного ключевого обмена, который завершается формированием общего сеансового ключа.

Хеш-функцией называется односторонняя функция, предназначенная для получения дайджеста или "отпечатков пальцев" файла, сообщения или некоторого блока данных.

История развития функций хеширования начинается с работ Картера, Вегмана, Симонсона, Биербрауера. Изначально функции хеширования использовались как функции создания уникального образа информационных последовательностей произвольной длины, с целью идентификации и определения их подлинности. Сам образ должен быть небольшим блоком фиксированной длины, как правило, 30, 60, 64, 128, 256, или 512 бит. Поэтому операции поиска сортировки и другие с большими массивами или базами данных существенно упрощаются, т.е. занимают гораздо меньшее время. Для обеспечения требуемой вероятности ошибки необходимо обеспечивать ряд требований к функции хеширования.

Электронно-цифровая подпись (**ЭЦП**) используется физическими и юридическими лицами в качестве аналога собственноручной подписи для придания электронному документу юридической силы, равной юридической силе документа на бумажном носителе, подписанного собственноручной подписью правомочного лица и скрепленного печатью.
Электронный документ - это любой документ, созданный и хранящийся на компьютере, будь то письмо, контракт или финансовый документ, схема, чертеж, рисунок или фотография.
ЭЦП - это программно-криптографическое средство, которое обеспечивает:

- проверку целостности документов;
- конфиденциальность документов;

установление лица, отправившего документ\

Билет №27

Архитектура программного обеспечения — это структура программы или вычислительной системы, которая включает программные компоненты, видимые снаружи свойства этих компонентов, а также отношения между ними. Этот термин также относится к документированию архитектуры программного обеспечения. Документирование архитектуры ПО упрощает процесс коммуникации между заинтересованными лицами, позволяет зафиксировать принятые на ранних этапах проектирования решения о высокоуровневом дизайне системы и позволяет использовать компоненты этого дизайна и шаблоны повторно в других проектах.

Языки описания архитектуры

Языки описания архитектуры (ADLS) используются для описания архитектуры программного обеспечения. Различными организациями было разработано несколько различных ADLS, в том числе AADL (стандарт SAE), Wright (разработан в университете Carnegie Mellon), Acme (разработан в университете Carnegie Mellon), xADL (разработан в UCI), Darwin (разработан в Imperial College в Лондоне), DAOP-ADL (разработан в Университете Малаги), а также ByADL (Университет L'Aquila, Италия). Общими элементами для всех этих языков являются понятия компонента, коннектора и конфигурации.

Виды (views)

Архитектура ПО обычно содержит несколько видов, которые аналогичны различным типам чертежей в строительстве зданий. В онтологии, установленной ANSI / IEEE 1471—2000, виды являются экземплярами точки зрения, где точка зрения существует для описания архитектуры с точки зрения заданного множества заинтересованных лиц.

Примеры видов:

* Функциональный/логический вид
* Вид код/модуль
* Вид разработки (development)/структурный
* Вид параллельности выполнения/процесс/поток
* Физический вид/вид развертывания
* Вид с точки зрения действий пользователя
* Вид с точки зрения данных

Хотя было разработано несколько языков для описания архитектуры программного обеспечения, но в настоящий момент нет согласия по поводу того, какой набор видов должен быть принят в качестве эталона. В качестве стандарта «для моделирования программных систем (и не только)» был создан язык UML.

Базовые фреймворки для архитектуры ПО (software architecture frameworks)

Существуют следующие фреймворки, относящихся к области архитектуры ПО:

- 4+1
- RM-ODP (Reference Model of Open Distributed Processing)

- Service-Oriented Modeling Framework (SOMF)

Такие примеры архитектур как фреймворк Захмана (Zachman Framework), DODAF и TOGAF относятся к области архитектуры предприятия (enterprise architectures).

Одна из моделей взаимодействия компьютеров в сети получила название «клиент-сервер». Каждый из составляющих эту архитектуру элементов играет свою роль: сервер владеет и распоряжается информационными ресурсами системы, клиент имеет возможность воспользоваться ими.

Сервер базы данных представляет собой мультипользовательскую версию СУБД, параллельно обрабатывающую запросы, поступившие со всех рабочих станций. В его задачу входит реализация логики обработки транзакций с применением необходимой техники синхронизации - поддержки протоколов блокирования ресурсов, обеспечение, предотвращение и/или устранения тупиковых ситуаций.

В ответ на пользовательский запрос рабочая станция получит не «сырье» для последующей обработки, а готовые результаты. Программное обеспечение рабочей станции при такой архитектуре играет роль только внешнего интерфейса (Front - end) централизованной системы управления данными. Это позволяет существенно уменьшить сетевой трафик, сократить время на ожидание блокированных ресурсов данных в мультипользовательском режиме, разгрузить рабочие станции и при достаточно мощной центральной машине использовать для них более дешевое оборудование.

Как правило, клиент и сервер территориально отделены друг от друга, и в этом случае они входят в состав или образуют систему распределенной обработки данных.

Для современных СУБД архитектура «клиент-сервер» стала фактически стандартом. Если предполагается, что проектируемая информация будет иметь архитектуру «клиент-сервер», то это означает, что прикладные программы, реализованные в ее рамках, будут иметь распределенный характер, т. е. часть функций приложений будет реализована в программе-клиенте, другая - в программе-сервере. Основной принцип технологии «клиент-сервер» заключается в разделении функций стандартного интерактивного приложения на четыре группы:

- функции ввода и отображения данных;
- прикладные функции, характерные для предметной области;
- фундаментальные функции хранения и управления ресурсами (базами данных);
- служебные функции.

АРХИТЕКТУРЫ МНОГОПРОЦЕССОРНЫХ СИСТЕМ

Основной характеристикой при классификации многопроцессорных вычислительных систем является способ организации оперативной памяти. В случае наличия общей памяти с равноправным доступом к ней от всех процессоров говорят о симметричных мультипроцессорных системах (SMP), а при использовании распределенной памяти, когда каждый процессор снабжается собственной локальной памятью, и прямой доступ к памяти других процессоров невозможен, речь идет о системах с массовым параллелизмом (MPP). Нечто среднее между SMP и MPP представляют собой NUMA-архитектуры (Non Uniform Memory Access), в которых память физически распределена, но логически общедоступна. При этом время доступа к различным блокам памяти становится неодинаковым. В одной из первых систем этого типа Cray T3D время доступа к памяти другого процессора было в 6 раз

больше, чем к своей собственной.В настоящее время развитие высокопроизводительных вычислительных систем идет по четырем основным направлениям: векторно-конвейерные суперкомпьютеры, SMP системы, MPP системы и кластерные системы.

распределенных систем

Одной из первых парадигм построения распределенных систем, был подход *удаленного вызова процедур* RPC (Remote procedure call). Отличительной особенностью RPC семейства, является высочайшая степень прозрачности, достигающаяся за счет подстановок средой исполнения соответствующих клиентских и серверных заглушек, а также максимально полной информации об используемых типах данных. С развитием ООП, на базе RPC были разработаны технологии получившие название *Распределенных Объектов* (Distributed objects). Основным их отличием от RPC, является использование *заместителя* (proxy), реализующего интерфейс удаленного серверного объекта. Назначение *заместителя* во многом аналогично RPC заглушке, это маршалинг аргументов при обращении к методам удаленного объекта, посылка сообщения удаленному объекту и демаршалинг результатов из ответных сообщений с последующим возвращением их клиенту[1].

Существует достаточно большое количество технологий реализующих парадигму распределенных объектов, наиболее распространенными из которых являются:

- .NET Remoting - .NET Remoting
- DCOM - Distributed Component Object Model
- CORBA - Common Object Request Broker Architecture
- RMI - Remote Method Invocation

Объе́ктно-ориенти́рованное, или **объектное**, программи́рование (в дальнейшем ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов. В случае языков с прототипированием вместо классов используются объекты-прототипы. В центре ООП находится понятие объекта. Объект — это сущность, которой можно посылать сообщения, и которая может на них реагировать, используя свои данные. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличиенаследования.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм; то есть возможность объектов с одинаковой спецификацией иметь различную реализацию.

Язык Self, соблюдая многие исходные положения объектно-ориентированного программирования, ввёл альтернативное классам понятие *прототипа*, положив началопрототипному программированию, считающемуся подвидом объектного.

Шаблоны проектирования

В разработке программного обеспечения, **шаблон проектирования** или **паттерн** — повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных

ситуациях. Объектно-ориентированные шаблоны
показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные.

На наивысшем уровне существуют **архитектурные шаблоны**, они охватывают собой архитектуру всей программной системы.

Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи.

Польза

Главная польза каждого отдельного шаблона состоит в том, что он описывает решение целого класса абстрактных проблем. Также тот факт, что каждый шаблон имеет свое имя, облегчает дискуссию об абстрактных структурах данных (ADT) между разработчиками, так как они могут ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация терминологии, названий модулей и элементов проекта.

Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова.

Проектирование пользовательского интерфейса

На практике высокоуровневое проектирование пользовательского интерфейса предваряет первоначальное проектирование, которое позволяет выявить требуемую функциональность создаваемого приложения, а также особенности его потенциальных пользователей. Указанные сведения можно получить, анализируя информацию, поступающую от пользователей. С этой целью производят опрос целевой аудитории и формируют профили пользователей. Профилями называют описания главных категорий пользователей. Одна из таких категорий может быть принята за основной профиль. Следует отметить, что набор характеристик, подробно описывающий пользователя, зависит от предметной области и контекста решаемых им задач. Поэтому работа по определению целей и задач пользователей и работа по формированию их профилей ведется параллельно.

Наиболее общий шаблон профиля содержит в себе следующие разделы:

- социальные характеристики;
- навыки и умения работы с компьютером;
- мотивационно-целевая среда;
- рабочая среда;
- особенности взаимодействия с компьютером (специфические требования пользователей, необходимые информационные технологии и др.).

Профили пользователей могут по необходимости расширяться за счет добавления других (значимых с точки зрения проектировщика) характеристик пользователей.

После выделения одного или нескольких основных профилей пользователей и после определения целей и задач, стоящих перед ними, переходят к следующему этапу проектирования. Этот этап связан с составлением пользовательских сценариев. Как правило, начинают с персонификации профилей (присваивания каждому профилю условного имени), затем формулируют сценарии. Сценарий - это описание действий, выполняемых пользователем в рамках решения конкретной задачи на пути достижения его цели. Очевидно, что достигнуть некоторой цели можно, решая ряд задач. Каждую их них пользователь может решать несколькими способами, следовательно, должно быть

сформировано несколько сценариев. Чем больше их будет, тем ниже вероятность того, что некоторые ключевые объекты и операции будут упущены.

В то же время, у разработчика имеется информация, необходимая для формализации функциональности приложения. А после формирования сценариев становится известным перечень отдельных функций. В приложении функция представлена функциональным блоком с соответствующей экранной формой (формами). Возможно, что несколько функций объединяются в один функциональный блок. Таким образом, на этом этапе устанавливается необходимое число экранных форм. Важно определить навигационные взаимосвязи функциональных блоков. На практике установлено наиболее подходящим число связей для одного блока равное трем. Иногда, когда последовательность выполнения функций жестко определена, между соответствующими функциональными блоками можно установить процессуальную связь. В этом случае их экранные формы вызываются последовательно одна из другой. Такие случаи имеют место не всегда, поэтому навигационные связи формируются либо исходя из логики обработки данных с которыми работает приложение, либо основываясь на представлениях пользователей (карточная сортировка). Навигационные связи между отдельными функциональными блоками отображаются на схеме навигационной системы. Возможности навигации в приложении передаются через различные навигационные элементы.

Основным навигационным элементом приложения является главное меню. Роль главного меню велика еще и потому, что оно осуществляет диалоговое взаимодействие в системе «пользователь-приложение». Кроме того, меню косвенно выполняет функцию обучения пользователя работе с приложением.

Формирование меню начинается с анализа функций приложения. Для этого в рамках каждой из них выделяют отдельные элементы: операции, выполняемые пользователями, и объекты, над которыми осуществляются эти операции. Следовательно, известно какие функциональные блоки должны позволять пользователю осуществлять какие операции над какими объектами. Выделение операций и объектов удобно проводить на основе пользовательских сценариев и функционала приложения. Выделенные элементы группируются в общие разделы главного меню. Группировка отдельных элементов происходит в соответствии с представлениями об их логической связи. Таким образом, главное меню может иметь каскадные меню, выпадающие при выборе какого либо раздела. Каскадное меню ставит в соответствие первичному разделу список подразделов.

Одним из требований к меню является их стандартизация, целью которой выступает формирование устойчивой пользовательской модели работы с приложением. Существуют требования, выдвигаемые с позиций стандартизации, которые касаются места размещения заголовков разделов, содержания разделов часто используемых в разных приложениях, формы заголовков, организации каскадных меню и др. Наиболее общие рекомендации стандартизации следующие:

- группы функционально связанных разделов отделяют разделителями (черта или пустое место);
- не используют в названиях разделов фраз (желательно не больше 2 слов);
- названия разделов начинают с заглавной буквы;
- названия разделов меню, связанных с вызовом диалоговых окон заканчивают многоточием;
- названия разделов меню, к которым относятся каскадные меню, заканчивают стрелкой;
- используют клавиши быстрого доступа к отдельным разделам меню. Их выделяют подчеркиванием;
- допускают использовать «горячие клавиши», соответствующие комбинации клавиш отображают в заголовках разделов меню;
- допускают использовать включение в меню пиктограмм;

- измененным цветом показывают недоступность некоторых разделов меню в ходе работы с приложением;
- допускают делать недоступные разделы невидимыми.

Билет №28

UML (англ. *Unified Modeling Language* — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения. UML является языком широкого профиля, это открытый стандарт, использующий графические обозначения для создания абстрактной моделисистемы, называемой *UML-моделью*. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но в средствах выполнения UML-моделей как интерпретируемого кода возможна кодогенерация.

Использование

Использование UML не ограничивается моделированием программного обеспечения. Его также используют для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML позволяет также разработчикам программного обеспечения достигнуть соглашения в графических обозначениях для представления общих понятий (таких как класс, компонент, обобщение (generalization), объединение (aggregation) и поведение), и больше сконцентрироваться на проектировании и архитектуре.

Диаграммы классов являются центральным звеном методологии объектно-ориентированных анализа и проектирования.

Диаграмма классов показывает классы и их отношения, тем самым представляя логический аспект проекта. Отдельная диаграмма классов представляет определенный ракурс структуры классов. На стадии анализа диаграммы классов используются, чтобы выделить общие роли и обязанности сущностей, обеспечивающих требуемое поведение системы. На стадии проектирования диаграммы классов используются, чтобы передать структуру классов, формирующих архитектуру системы.

Каждый класс должен иметь имя; если имя слишком длинно, его можно сократить или увеличить сам значок на диаграмме. Имя каждого класса должно быть уникально в содержащем его проекте.

Диаграмма классов определяет типы объектов системы и различного рода статические связи, которые существуют между ними. Имеется два основных вида статических связей:

- ассоциации (например, менеджер может вести несколько проектов),
- подтипы (работник является разновидностью личности).

На диаграммах классов изображаются также атрибуты классов, операции и ограничения, которые накладываются на связи между объектами. На рис. 11.1 изображена типичная диаграмма классов. Далее будут рассмотрены различные фрагменты диаграммы.

Ассоциации

Ассоциации представляют собой связи между экземплярами классов (личность работает в компании, компания имеет ряд офисов).

Атрибуты

Атрибуты во многом подобны ассоциациям. Разница между ними заключается в том, что атрибуты предполагают единственное направление навигации - от типа к атрибуту.

Операции

Операции представляют собой процессы, реализуемые классом. Наиболее очевидное соответствие существует между операциями и методами над классом.

Полный синтаксис UML для операций выглядит следующим образом: <признак видимости> <имя> (<список-параметров>): <тип-выражения-возвращающего-значение> = <строка-свойств>, где

- признак видимости может принимать те же значения, что и для атрибутов;
- имя представляет собой символьную строку;
- список-параметров содержит необязательные аргументы, синтаксис которых совпадает с синтаксисом атрибутов;
- тип-выражения-возвращающего-значение является необязательной спецификацией и зависит от конкретного языка программирования;
- строка-свойств показывает значения свойств, которые применяются к данной операции

Обобщение

Смысл обобщения заключается в том, что интерфейс подтипа должен включать все элементы интерфейса супертипа. Другая сторона обобщения связана с принципом подстановочности.

Диаграммы вариантов использования (use-case diagrams)

Одна из моделей формализации процесса постановки целей и задач проекта была предложена фирмой Rational и вошла в стандарт языка UML. Для этого применяются диаграммы вариантов использования (use-case), иногда называемые диаграммами прецедентов. Вариант использования представляет собой типичное взаимодействие пользователя и проектируемой системы. Варианты использования характеризуются рядом свойств:

- вариант использования охватывает некоторую очевидную для пользователей функцию;
- вариант использования может быть как небольшим, так и достаточно крупным;
- вариант использования решает некоторую дискретную задачу пользователя.

В простейшем случае вариант использования создается в процессе обсуждения с пользователями тех вещей, которые они хотели бы получить от системы. При этом каждой отдельной функции, которую они хотели бы реализовать, присваивается некоторое имя и записывается ее краткое текстовое описание.

Это все, что необходимо в фазе анализа. Знание некоторых деталей может потребоваться, если предполагается, что данный вариант использования содержит важные архитектурные ответвления. Большинство вариантов использования может быть детализировано во время конкретной итерации в процессе проектирования.

Действующие лица могут играть различные роли по отношению к варианту использования. Они могут применять его результаты или сами непосредственно в нем участвовать.

Хорошим источником для идентификации вариантов использования служат внешние события. Для этого необходимо перечислить все происходящие во внешнем мире события, на которые система должна реагировать. Какое-либо конкретное событие может повлечь за собой реакцию системы, не требующую вмешательства пользователей, или, наоборот,

вызвать чисто пользовательскую реакцию. Идентификация событий, на которые необходимо реагировать, поможет идентифицировать варианты использования.

Билет №29

Сегодня веб-сайт стал привычной стороной деятельности современного человека. В настоящий момент веб-сайты действуют у крупных холдингов и частных коммерческих палаток. Иногда в сети Интернет можно найти частные веб-сайты, созданные на высоком профессиональном уровне. При этом потенциальную популярность веб-сайта у посетителей определяет технология разработки web-приложений, используемых при создании.

Принципы действия технологий разработки веб-приложений

В настоящий момент действуют различные программы, на базе которых проводится разработка веб-сайтов. Каждая из них используется для создания различных интернет-продуктов. Ведь современные веб-сайты – живой и динамично меняющихся организм. В результате технологии разработки веб-приложений должны обязательно подразумевать компонентную структуру front- и back-office, дающую возможность быстро добавлять новый контент. Желательно чтобы код был совместим с промышленными кодами приложений.

Технология должна подразумевать возможность работы на разных языках и поддерживать разнообразные входные форматы. Также обычно требуется поддержка тонкого разграничения доступа

Hyperlink (Гиперссылка)
Графическое изображение или текст на сайте (или в письме электронной почты, устанавливающие связь и позволяющие переходить к другим объектам сети Интернет. Это строка в HTML-документе, указывающая на другой файл, который может быть расположен в сетиИнтернет, и содержащая полный путь к этому файлу. Чтобы отличить их от обычного текста, гиперссылки обычно выделяются жирным шрифтом или подчеркиванием.

CSS (Cascading Style Sheets)
Язык иерархических стилевых спецификаций. Был разработан как дополнение к HTML, с целью восполнить ограниченные возможности этого языка в области визуального форматирования. Позволяет описать форматирование всего документа или группы документов, а также добавлять некоторые эффекты изображениям и тексту (например, изменение цвета ссылки при наведении мышкой). Помогает иногда значительно сократить размеры документов. Работает только в броузерах версий 4.0 и старше.

Frames (Фреймы)
Элемент языка HTML, позволяющий жестко разделить страницу на несколько независимых окон и в каждом из них размещать свою собственную WEB-страничку. Возможна ссылка из одного окна в другое. Применяется в основном для организации постоянно находящихся на экране меню, в то время как в другом окне располагается непосредственно сама информация.

ASP (Active Server Pages)
Смесь средств программирования с использованием языка HTML,

выполняющих чтение и запись в базу данных посредством ODBC. Предоставляет массу возможностей, не требующих применения, но похожих на такие средства программирования, как CGI, JavaScript, Perl, ActiveX и ISAPI. Все задействованные ASP-страницами программы запускаются и выполняются на сервере, причем броузер получает только результирующие HTML-файлы.

Flash (Флэш)

Технология Shockwave / Flash, которую разработала Macromedia Inc. для того, чтобы разнообразить обычные текстовые страницы веб-сайта красочной и интерактивной векторной графикой. Технология Flash позволяет создавать как поражающие воображение презентационные ролики, так и интерактивные интерфейсы, создающие новое качество комфорта на сайте.

Script (Скрипт)

Программа, написанная на каком-либо языке программирования для взаимодействия клиента с сервером. Например: Script на Perl для подсчета количества посещений.

JavaScript (Ява-скрипт)

Фирма Netscape разработала в 1995 году замечательный инструмент, позволяющий HTML-странице, загруженной в браузер, динамически управлять своим содержимым и самим браузером. Последняя реализация JavaScript, называемая "динамический HTML" позволяет реализовать на веб-странице почти полноценный пользовательский интерфейс с выпадающими многоуровневыми меню, перетаскиванием объектов мышью, анимацией и т.п. Текст программы встроен непосредственно в HTML-документ и интерпретируется самим броузером. Применяется в основном для создания таких эффектов, как: бегущая строка, рисунки, изменяющие свой вид при подведении курсора и т.д.

PHP

Язык программирования, дающий возможность легко и быстро создавать динамично изменяемые html-страницы, профессионального программирования для Интернет, создания профессиональных web-сайтов и web-приложений.

Perl

Язык программирования. Программы, написанные на Perl, запускаются на стороне сервера. В основном применяется на UNIX-ориентированных WEB-серверах. Применяется для обеспечения доступа к базам данным, создания динамических страничек и т.п.

Rails — это полноценный, многоуровневый фреймворк для построения веб-приложений, использующих базы данных, который основан на архитектуре Модель-Представление-Контроллер (Model-View-Controller, MVC). Динамичный AJAX-интерфейс, обработка запросов и выдача данных в контроллерах, предметная область, отраженная в базе данных, — для всего этого Rails предоставляет однородную среду разработки на Ruby. Все, что необходимо для начала — база данных и веб-сервер.Rails используются всеми кампаниями — от стартапов и некоммерческих организаций до крупного бизнеса. Rails — это прежде всего инфраструктура, поэтому среда великолепно подходит для любого типа веб-приложений, будь это программы для организации совместной работы, поддержки сообществ, электронного бизнеса, управления содержанием, статистики, управления.Rails отлично работает со многими веб-серверами и СУБД. В качестве веб-сервера рекомендуется использовать Apache или nginx с модулем Phusion

Passenger. Rails также можно разворачивать используя Unicorn, Thin, Mongrel или FastCGI. В качестве СУБД можно использоват MySQL, PostgreSQL, SQLite, Oracle, SQLServer, DB2 или Firebird. Использовать Rails можно на практически любой операционной системе, однако для развертывания рекомендуется системы семейства *nix.

jQuery — это сборник функций и разновидностей абстрактного типа данных, который характерен особенностью построения на языке JavaScript, основанный на его взаимодействии с HTML. С помощью jQuery можно легко получить доступ к DOM (Document Object Model) и манипулировать его элементами. Также jQuery обеспечивает удобный интерфейс программирования приложений (API) по работе с асинхронным JavaScript и XML (AJAX).

Система управления версиями — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (*Software Configuration Management Tools*).

XML — рекомендованный Консорциумом Всемирной паутины язык разметки, фактически представляющий собой свод общих синтаксических правил. XML — текстовый формат, предназначенный для хранения структурированных данных (взамен существующих файлов баз данных), для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки (например, XHTML). XML является упрощённым подмножеством языка SGML.

JSON — текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком. Как и многие другие текстовые форматы, JSON легко читается людьми.

Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается языконезависимым и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

YAML — человекочитаемый формат сериализации данных, концептуально близкий к языкам разметки, но ориентированный на удобство ввода-вывода типичных структур данных многих языков программирования.

Название YAML представляет собой рекурсивный акроним *YAML Ain't Markup Language* («YAML — не язык разметки»). В названии отражена история развития: на ранних этапах язык назывался *Yet Another Markup Language* («Ещё один язык разметки») и даже рассматривался как конкурент XM, но позже был переименован с целью акцентировать внимание на данных, а не на разметке документов.

Билет №30

При обмене данными по каналам связи используются три метода передачи данных:

1) Симплексная (однонаправленная) — TV, радио;

2) Полудуплексная передача — (приём и передача данных осуществляются поочерёдно);

3) Дуплексная (двунаправленная) – каждая станция одновременно передаёт и принимает данные.

Для передачи данных в информационных системах наиболее часто применяется последовательная (полудуплексная) передача. Она разделяется на два метода:

а) Асинхронная передача;

б) Синхронная передача.

При асинхронной передаче каждый символ передаётся отдельной посылкой. Стартовые биты предупреждают о начале передачи. Затем передаётся символ. Для определения достоверности передачи используется бит чётности (бит чётности равен 1, если количество единиц в символе нечётно, и равен 0 в противном случае). Последний бит сигнализирует об окончании передачи.

Преимущества:

1) Несложная отработанная система;

2) Недорогое интерфейсное оборудование.

Недостатки:

1) Третья часть пропускной способности теряется на передачу служебных битов;

2) Невысокая скорость передачи данных по сравнению с синхронной;

3) При множественной ошибке с помощью бита чётности невозможно определить достоверность полученной информации.

Асинхронная передача используется в системах, где обмен данными происходит время от времени, и не требуется высокая скорость передачи данных.

При использовании синхронного метода данные передаются блоками. Для синхронизации работы приёмника и передатчика в начале блока передаются биты синхронизации. Затем передаются данные, код обнаружения ошибки и символ окончания передачи. Код обнаружения ошибки вычисляется по содержимому поля данных и позволяет однозначно определить достоверность принятой информации.

Преимущества:

1) Высокая эффективность передачи данных;

2) Высокая скорость передачи данных;

3) Надёжный встроенный механизм обнаружения ошибок.

Недостатки:

1) Интерфейсное оборудование более сложное и дорогое.

Обмен данными через канал прямого доступа к памяти

Этот режим в большинстве операционных систем и BIOS не предусмотрен, однако стандартом ATA-2 он поддерживается. Передача через канал прямого доступа к памяти (DMA) означает, что, в отличие от режима РЮ, данные передаются непосредственно из жесткого диска в системную (основную) память, минуя центральный процессор.

Прямой доступ к памяти может осуществляться двумя способами: обычным и режиме Bus Master. В первом случае обработка запросов, захват шины и передача данных осуществляются контроллером DMA на системной плате. Во втором случае все эти операции выполняет устройство, смонтированное на самой плате интерфейса. Это, естественно, увеличивает сложность и стоимость интерфейсов подобного типа.

В системах с микросхемой Intel PIIX (PCI IDE ISA eXcelerator) и более поздними компоненты South Bridge могут поддерживать режим Bus Master IDE. При этом используется режим Bus Master на шине PCI при передаче данных. Режимы Bus Master IDE и скорости передачи приведены в табл. 7.5.

К сожалению, даже самый быстрый режим Bus Master IDE 2 имеет ту же скорость передачи 16,67 Мбайт/с, что и режим РЮ 4. Это связано с тем, что контроллеры DMA в компьютерах с шиной ISA обладают очень низким быстродействием. И поэтому нет никакого смысла использовать их для работы с современными жесткими дисками. В большинстве случаев рекомендуется использовать стандартный режим РЮ 4, если дисководы его поддерживают. Режимы Bus Master IDE никогда не были очень эффективными и теперь заменены режимами Ultra-DMA, поддерживаемыми совместимыми устройствами ATA-4.

Стандартные периферийные интерфейсы

1 Интерфейс SCSI

SCSI – интерфейс, разработанный для объединения на одной шине различных по своему назначению устройств, таких как жёсткие диски, накопители на магнитооптических дисках, приводы CD, DVD, стримеры, сканеры, принтеры и т. д. Раньше имел неофициальное название Shugart Computer Systems Interface в честь создателя Алана Ф. Шугарта.

Существует три стандарта SCSI:

1. SE (single-ended) – асимметричный SCSI, для передачи каждого сигнала используется отдельный проводник.
2. LVD (low-voltage-differential) – интерфейс дифференциальной шины низкого напряжения, сигналы положительной и отрицательной полярности идут по разным физическим проводам. На один сигнал приходится по одной витой паре проводников. Используемое напряжение при передаче сигналов +1,8 В.
3. HVD (high-voltage-differential) – интерфейс дифференциальной шины высокого напряжения, отличается от LVD повышенным напряжением и

специальными приемопередатчиками.

2 Интерфейс SAS

Serial Attached SCSI (SAS) – компьютерный интерфейс, разработанный для обмена данными с такими устройствами, как жёсткие диски, накопители на оптическом диске и т.д. SAS использует последовательный интерфейс для работы с непосредственно подключаемыми накопителями (Direct Attached Storage (DAS) devices). SAS разработан для замены параллельного интерфейса SCSI и позволяет достичь более высокой пропускной способности, чем SCSI; в то же время SAS совместим с интерфейсом SATA. Хотя SAS использует последовательный интерфейс в отличие от параллельного интерфейса, используемого традиционным SCSI, для управления SAS-устройствами по-прежнему используются команды SCSI. Протокол SAS разработан и поддерживается комитетом T10. SAS поддерживает передачу информации со скоростью до 3 Гбит/с. Благодаря уменьшенному разъему SAS обеспечивает полное двухпортовое подключение как для 3,5-дюймовых, так и для 2,5-дюймовых дисковых накопителей (раньше эта функция была доступна только для 3,5-дюймовых дисковых накопителей с интерфейсом Fibre Channel).

Интерфейс IDE предназначен для подключения устройств хранения данных, обладающих собственным контроллером. В настоящее время интерфейс ATA/ AT API является самым массовым интерфейсом устройств хранения данных, причем не только в мире PC-совместимых компьютеров. Пока что наибольшее распространение получил его «классический» параллельный вариант, ему на смену идут последовательные интерфейсы Serial ATA (SATA) и Serial ATA-II (SATA-II). Теперь параллельный интерфейс ATA/ATAPI стали называть PATA (Parallel ATA — параллельный интерфейс ATA). Параллельный интерфейс ATA (Advanced Technology Attachment) был введен в конце 1980-х годов как интерфейс для подключения накопителей на жестких магнитных дисках к компьютерам IBM PC AT с шиной ISA. Интерфейс появился в результате переноса стандартного (для PC/AT) контроллера накопителя на жестком диске (Hard Disc Controller, HDC) ближе к накопителю, то есть создания устройств со встроенным контроллером (Integrated Drive Electronics, IDE). Для связи устройства с системной шиной ISA использовали лен точный кабель с параллельным шинным интерфейсом, получившим названия ATA и IDE, которые, фактически, являются синонимами. В этом интерфейсе используются сигналы шины ISA, часть из которых буферизовали на небольшой плате адаптера IDE, устанавливаемого в слот ISA, а часть направили прямо на разъем нового интерфейса.

(книга, посвященная тематики http://window.edu.ru/library/pdf2txt/751/72751/50478/page15)

Билет №31

Конвейерная обработка - способ выполнения команд процессором, при котором выполнение следующей команды начинается до полного окончания выполнения предыдущей команды (в предположении отсутствия ветвления).
Возможность конвейерной обработки связана с разделением процесса выполнения команд на последовательные этапы: выборки команды, дешифровки, выборки операндов, выполнения команды, запись результата в память.
Векторная обработка - единообразная обработка последовательностей данных, встречающаяся, как правило, при манипулировании матрицами (элементами которых являются векторы) или другими информационными массивами. *Векторная обработка* в МКП осуществляется итеративными операциями над наборами упорядоченных данных - векторами, размещенными в локальной памяти.
Средства *векторной обработки* каждого центрального процессора включают в себя конвейерное арифметическое устройство и набор из 16 векторных регистров по 128 32-битных элементов в каждом. Эти средства реализуют 171 векторную команду.
Для поддержки *конвейерной и векторной обработки* необходимы быстрые регистровые средства, освобождающие от необходимости обращения к памяти при выполнении векторных операций и обеспечивающие высокий темп загрузки конвейеров и информационные связи между ними.

Основным параметром классификации паралелльных компьютеров является наличие общей (SMP) или распределенной памяти (MPP). Нечто среднее между SMP и MPP представляют собой NUMA-архитектуры, где память физически распределена, но логически общедоступна. Кластерные системы являются более дешевым вариантом MPP.
При поддержке команд обработки векторных данных говорят о векторно-конвейерных процессорах, которые, в свою очередь могут объединяться в PVP-системы с использованием общей или распределенной памяти. Все большую популярность приобретают идеи комбинирования различных архитектур в одной системе и построения неоднородных систем.
При организациях распределенных вычислений в глобальных сетях (Интернет) говорят о мета-компьютерах, которые, строго говоря, не представляют из себя параллельных архитектур.

Массивно-параллельные системы (MPP)

Архитектура	Система состоит из однородных *вычислительных узлов*, включающих: • один или несколько центральных процессоров (обычно RISC), • локальную память (прямой доступ к памяти других узлов невозможен),

	• коммуникационный процессор или сетевой адаптер • иногда - жесткие диски (как в SP) и/или другие устройства В/В К системе могут быть добавлены специальные узлы ввода-вывода и управляющие узлы. Узлы связаны через некоторую коммуникационную среду (высокоскоростная сеть, коммутатор и т.п.)
Примеры	IBM RS/6000 SP2, Intel PARAGON/ASCI Red, CRAY T3E, Hitachi SR8000, транспьютерные системы Parsytec.
Масштабируемость	Общее число процессоров в реальных системах достигает нескольких тысяч (ASCI Red, Blue Mountain).
Операционная система	Существуют два основных варианта: 1. Полноценная ОС работает только на управляющей машине (front-end), на каждом узле работает сильно урезанный вариант ОС, обеспечивающие только работу расположенной в нем ветви параллельного приложения. Пример: Cray T3E. 2. На каждом узле работает полноценная UNIX-подобная ОС (вариант, близкий к кластерному подходу). Пример: IBM RS/6000 SP + ОС AIX, устанавливаемая отдельно на каждом узле.
Модель программирования	Программирование в рамках модели **передачи сообщений** (MPI, PVM, BSPlib)

Симметричные мультипроцессорные системы (SMP)

Архитектура	Система состоит из нескольких однородных процессоров и массива общей памяти (обычно из нескольких независимых блоков). Все процессоры имеют доступ к любой точке памяти с одинаковой скоростью. Процессоры подключены к памяти либо с помощью общей шины (базовые 2-4 процессорные SMP-сервера), либо с помощью crossbar-коммутатора (HP 9000). Аппаратно поддерживается когерентность кэшей.
Примеры	HP 9000 V-class, N-class; SMP-сервера и рабочие станции на базе процессоров Intel (IBM, HP, Compaq, Dell, ALR, Unisys, DG, Fujitsu и др.).
Масштабируемость	Наличие общей памяти сильно упрощает взаимодействие процессоров между собой, однако накладывает сильные ограничения на их число - не более 32 в реальных системах. Для построения масштабируемых систем на базе SMP используются кластерныеили NUMA-архитектуры.
Операционная	Вся система работает под управлением единой ОС (обычно

система	UNIX-подобной, но для Intel-платформ поддерживается Windows NT). ОС автоматически (в процессе работы) распределяет процессы/нити по процессорам (scheduling), но иногда возможна и явная привязка.
Модель программирования	Программирование в модели **общей памяти**. (POSIX threads, OpenMP). Для SMP-систем существуют сравнительно эффективные средства автоматического распараллеливания.

Системы с неоднородным доступом к памяти (NUMA)

Архитектура	Система состоит из однородных базовых модулей (плат), состоящих из небольшого числа процессоров и блока памяти. Модули объединены с помощью высокоскоростного коммутатора. Поддерживается единое адресное пространство, аппаратно поддерживается доступ к удаленной памяти, т.е. к памяти других модулей. При этом доступ к локальной памяти в несколько раз быстрее, чем к удаленной. В случае, если аппаратно поддерживается когерентность кэшей во всей системе (обычно это так), говорят об архитектуре **cc-NUMA** (cache-coherent NUMA)
Примеры	HP HP 9000 V-class в SCA-конфигурациях, SGI Origin2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000, SNI RM600.
Масштабируемость	Масштабируемость NUMA-систем ограничивается объемом адресного пространства, возможностями аппаратуры поддежки когерентности кэшей и возможностями операционной системы по управлению большим числом процессоров. На настоящий момент, максимальное число процессоров в NUMA-системах составляет 256 (Origin2000).
Операционная система	Обычно вся система работает под управлением единой ОС, как в SMP. Но возможны также варианты динамического "подразделения" системы, когда отдельные "разделы" системы работают под управлением разных ОС (например, Windows NT и UNIX в NUMA-Q 2000).
Модель программирования	Аналогично SMP.

Параллельные векторные системы (PVP)

Архитектура	Основным признаком PVP-систем является наличие специальных векторно-конвейерных процессоров, в которых предусмотрены команды однотипной обработки векторов независимых данных, эффективно выполняющиеся на конвейерных функциональных устройствах.

	Как правило, несколько таких процессоров (1-16) работают одновременно над общей памятью (аналогично SMP) в рамках многопроцессорных конфигураций. Несколько таких узлов могут быть объединены с помощью коммутатора (аналогично MPP).
Примеры	NEC SX-4/SX-5, линия векторно-конвейерных компьютеров CRAY: от CRAY-1, CRAY J90/T90, CRAY SV1, CRAY X1, серия FujitsuVPP.
Модель программирования	Эффективное программирование подразумевает векторизацию циклов (для достижения разумной производительности одного процессора) и их распараллеливание (для одновременной загрузки нескольких процессоров одним приложением).

Кластерные системы

Архитектура	Набор рабочих станций (или даже ПК) общего назначения, используется в качестве дешевого варианта массивно-параллельногокомпьютера. Для связи узлов используется одна из стандартных сетевых технологий (Fast/Gigabit Ethernet, Myrinet) на базе шинной архитектуры или коммутатора. При объединении в кластер компьютеров разной мощности или разной архитектуры, говорят о **гетерогенных** (неоднородных) кластерах. Узлы кластера могут одновременно использоваться в качестве пользовательских рабочих станций. В случае, когда это не нужно, узлы могут быть существенно облегчены и/или установлены в стойку.
Примеры	NT-кластер в NCSA, Beowulf-кластеры.
Операционная система	Используются стандартные для рабочих станций ОС, чаще всего, свободно распространяемые - Linux/FreeBSD, вместе со специальными средствами поддержки параллельного программирования и распределения нагрузки.
Модель программирования	Программирование, как правило, в рамках модели передачи сообщений (чаще всего - MPI). Дешевизна подобных систем оборачивается большими накладными расходами на взаимодействие параллельных процессов между собой, что сильно сужает потенциальный класс решаемых задач.

Программно-аппаратная архитектура для вычислений на GPU компании NVIDIA отличается от предыдущих моделей GPGPU тем, что позволяет писать программы для GPU на настоящем языке Си со стандартным синтаксисом, указателями и необходимостью в минимуме расширений для доступа к вычислительным ресурсам видеочипов. CUDA не

зависит от графических API, и обладает некоторыми особенностями, предназначенными специально для вычислений общего назначения.

Преимущества CUDA перед традиционным подходом к GPGPU вычислениям:

- интерфейс программирования приложений CUDA основан на стандартном языке программирования Си с расширениями, что упрощает процесс изучения и внедрения архитектуры CUDA;
- CUDA обеспечивает доступ к разделяемой между потоками памяти размером в 16 Кб на мультипроцессор, которая может быть использована для организации кэша с широкой полосой пропускания, по сравнению с текстурными выборками;
- более эффективная передача данных между системной и видеопамятью
- отсутствие необходимости в графических API с избыточностью и накладными расходами;
- линейная адресация памяти, и gather и scatter, возможность записи по произвольным адресам;
- аппаратная поддержка целочисленных и битовых операций.

Основные ограничения CUDA:

- отсутствие поддержки рекурсии для выполняемых функций;
- минимальная ширина блока в 32 потока;
- закрытая архитектура CUDA, принадлежащая NVIDIA.

Основные преимущества CUDA по сравнению с предыдущими методами GPGPU вытекают из того, что эта архитектура спроектирована для эффективного использования неграфических вычислений на GPU и использует язык программирования C, не требуя переноса алгоритмов в удобный для концепции графического конвейера вид. CUDA предлагает новый путь вычислений на GPU, не использующий графические API, предлагающий произвольный доступ к памяти (scatter или gather). Такая архитектура лишена недостатков GPGPU и использует все исполнительные блоки, а также расширяет возможности за счёт целочисленной математики и операций битового сдвига.

CUDA использует параллельную модель вычислений, когда каждый из SIMD процессоров выполняет ту же инструкцию над разными элементами данных параллельно. GPU является вычислительным устройством, сопроцессором (device) для центрального процессора (host), обладающим собственной памятью и обрабатывающим параллельно большое количество потоков. Ядром (kernel) называется функция для GPU, исполняемая потоками (аналогия из 3D графики — шейдер).

Модель программирования в CUDA предполагает группирование потоков. Потоки объединяются в блоки потоков (thread block) — одномерные или двумерные сетки потоков, взаимодействующих между собой при помощи разделяемой памяти и точек синхронизации. Программа (ядро, kernel) исполняется над сеткой (grid) блоков потоков (thread blocks), см. рисунок ниже. Одновременно исполняется одна сетка. Каждый блок может быть одно-, двух- или трехмерным по форме, и может состоять из 512 потоков на текущем аппаратном обеспечении. Блоки потоков выполняются в виде небольших групп, называемых варп (warp), размер которых — 32 потока. Это минимальный объём данных, которые могут обрабатываться в мультипроцессорах. И так как это не всегда удобно, CUDA позволяет работать и с блоками, содержащими от 64 до 512 потоков. Группировка блоков в сетки позволяет уйти от ограничений и применить ядро к большему числу потоков за один вызов. Это помогает и при масштабировании. Если у GPU недостаточно ресурсов, он будет выполнять блоки последовательно. В обратном случае, блоки

могут выполняться параллельно, что важно для оптимального распределения работы на видеочипах разного уровня, начиная от мобильных и интегрированных.

БИЛЕТ №32

OpenMP (Open Multi-Processing) - это набор директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью (SMP-системах).

В OpenMP используется модель параллельного выполнения "ветвление-слияние". Программа OpenMP начинается как единственный поток выполнения, называемый начальным потоком. Когда поток встречает параллельную конструкцию, он создает новую группу потоков, состоящую из себя и некоторого числа дополнительных потоков, и становится главным в новой группе. Все члены новой группы (включая главный) выполняют код внутри параллельной конструкции. В конце параллельной конструкции имеется неявный барьер. После параллельной конструкции выполнение пользовательского кода продолжает только главный поток. В параллельный регион могут быть вложены другие параллельные регионы, в которых каждый поток первоначального региона становится основным для своей группы потоков. Вложенные регионы могут в свою очередь включать регионы более глубокого уровня вложенности.

MPI - это библиотека передачи сообщений, собрание функций на C/C++ (или подпрограмм в Фортране), облегчающих коммуникацию (обмен данными и синхронизацию задач) между процессами параллельной программы с распределенной памятью. MPI (сокращение по первым буквам) обозначает Message Passing Interface (интерфейс передачи сообщений). MPI является на данный момент фактическим стандартом и самой развитой переносимой библиотекой параллельного программирования с передачей сообщений.

T-система

OpenTS — это система параллельного программирования для всего спектра параллельных архитектур: многоядерные процессоры, SMP-установки, кластерные и распределённые вычислительные системы. OpenTS — оригинальная российская разработка, которая ведётся в Институте программных систем РАН в течении более чем 20 лет. В настоящее время система развивается в рамках суперкомпьютерной программы "СКИФ" и "СКИФ-ГРИД" Союзного государства России и Беларуси, а также других проектов.

OpenTS (T-система с открытой архитектурой) — это современная реализация идей T-системы. Она отличается от прочих разработок тем, что обеспечивает автоматическое динамическое распараллеливание программ и предоставляет среду исполнения для языка программирования высокого уровня T++, который является параллельным диалектом языка C++.

В T-системе использовано автоматическое динамическое распараллеливание. Фрагменты программы, которые могут быть выполнены параллельно, обнаруживаются системой (*операционной средой параллельного выполнения*) в динамике — в процессе выполнения программы.

Это позволяет проводить балансировку загрузки отдельных процессорных элементов, составляющих вычислительную систему и, как следствие, без изменения исполнять системные и прикладные программы на вычислительных установках различной конфигурации (*адаптируемость к изменению конфигурации*).

Б**а**зовые принципы T-системы основаны на результатах общей теории *функционального программирования (ФП)*. Математические функции выражают связь между параметрами (входом) и результатом (выходом) некоторого процесса. Вычисление – также процесс, имеющий вход и выход, при этом функция является вполне подходящим и адекватным средством описания вычислений.

Одно из глав**е**нствующих понятий T-системы – «*чистые» функции* (функции без *побочных эффектов* при вычислениях); чистая функция может быть выполнена параллельно с основной программой. В каждый момент времени выделяются готовые к вычислению «подвыражения» и распределяются по имеющимся процессорам, при этом за основу берется граф, узлы которого представляют выбранные функции, а дуги соответствуют отношению «выражение-подвыражение».

Для использования возможностей функционального программирования в традиционные языки вводится понятие *неготового значения*. В применении к C это выражается введением дополнительного атрибута tval описания переменных, tval int j определяет переменную, значение которой может быть целым числом или неготовым (пока не посчитанным) значением, в описании функции без побочных эффектов необходим атрибут tfun, выходного значения – tout, глобальной ссылки на неготовую переменную - tptr (измен**е**нный таким образом C++ получил название T++.

T-система запускает чистые функции как ветви параллельной программы, а *неготовые переменные* являются основным средством синхронизации их выполнения.

Основные ключевые слова языка T++:

 tval — атрибут, указываемый в определениях T-переменных, которые могут содержать "неготовые" значения;

 tptr — употребляется для описания "удаленного" указателя —указателя на T-объект (например, неготовое значение), который может находиться на другом узле кластера.

 tout — атрибут, указываемый в определениях T-функций у выходных результатов. Ключевое слово tout может встречаться только в описании аргументов T-функции.

 tfun — признак того, что определяемая функция может вычисляться параллельно (T-функция).T-функция не может являться методом, это должна быть обычная функция верхнего уровня.T-

система поддерживает коарность Т-функций — возможность возвращения Т-функцией нескольких результатов.