

ВОПРОСЫ К ВСТУПИТЕЛЬНОМУ ЭКЗАМЕНУ

1. Задачи поиска: исчерпывающий поиск, быстрый поиск, использование деревьев в задачах поиска.....	5
1.1. Задачи поиска.....	5
1.2. Исчерпывающий поиск: перебор с возвратом, метод ветвей и границ, динамическое программирование.....	5
1.3. Быстрый поиск: бинарный и последовательный поиски в массивах, хеширование.....	7
1.4. Использование деревьев в задачах поиска: бинарные и случайные бинарные, оптимальные и сбалансированные деревья поиска.....	7
2. Уровни моделей и этапы проектирования баз данных Инфологическое моделирование	10
1.1. Базы данных (БД) и системы управления базой данных (СУБД)	10
1.2. Выбор системы управления базами данных	12
1.3. Жизненный цикл базы данных	12
1.4. Уровни моделей и этапы проектирования БД.....	13
1.5. Инфологическое моделирование.....	15
1.6. Языковые средства современных СУБД.....	16
1.7. Даталогическое моделирование.....	16
1.8. Проектирование на физическом уровне.....	17
1.9. Средства и методы проектирование БД.....	17
1.10. Ограничения целостности.....	18
1.11. Технология оперативной обработки транзакции (OLTP-технология).....	19
1.12. Информационные хранилища	20
1.13. OLAP-технология	20
3. Принципы построения и архитектура компьютерных сетей Протоколы, иерархия протоколов и режимы их работы	22
1.1. Классификация современных компьютерных сетей	22
1.2. Принципы построения и архитектура компьютерных сетей	24
1.3. Протоколы, иерархия протоколов и режимы их работы	28
1.4. Соединение, передача данных, разъединение	30
1.5. Передача информации в компьютерных сетях.....	30
1.6. Каналы связи, модемы.....	31
1.7. Кодирование и защита от ошибок.....	32
1.8. Структура пакета	33
1.9. Методы коммутации каналов, сообщений, пакетов.....	35
1.10. Маршрутизация.....	40
1.11. Базовые средства передачи данных	40
1.12. Локальные вычислительные сети (ЛВС)	41
1.13. Структура и принципы строения ЛВС.....	41
1.14. Конфигурация связей.....	42
1.15. Стандарты, соглашения и рекомендации	44
1.16. Программное обеспечение компьютерных сетей.....	46
4. Назначение и основные функции операционных систем.....	48
4.1. Назначение и основные функции операционных систем (ОС).....	48
4.2. Способы построения современных операционных систем и операционных оболочек	56
4.3. Организация и управление памятью, распределение ресурсов, сервисные службы операционных систем, организация сохранности и защиты программных систем	59
5. Языки и системы программирования Модели языков программирования.....	62
5.1. Языки и системы программирования.....	62
5.2. Компиляторы и интерпретаторы	64
5.3. Объектно-ориентированное программирование	65
6. Модели и этапы разработки программного обеспечения	67
6.1. Программные средства и программные продукты	71
6.2. Коммерческое, условно-бесплатное и свободно распространяемое программное обеспечение.....	72

6.3.	Теория схем программ.....	74
6.4.	Семантическая теория программ.....	75
6.5.	Модели вычислительных процессов: Модель графов распределения ресурсов.....	76
6.6.	Вычислительные схемы	79
7.	Реляционные системы управления базами данных Объектно-ориентированные базы данных	80
7.1.	Реляционные СУБД.....	80
7.2.	СУБД на инвертированных файлах.....	83
7.3.	Гипертекстовые и мультимедийные БД.....	84
7.4.	XML-серверы	85
7.5.	Объектно-ориентированные базы данных.....	85
7.6.	Организация процессов обработки данных в БД	89
8.	Архитектуры вычислительных систем Архитектура системы команд.....	92
8.1.	Классификация современных вычислительных систем	95
8.2.	Способы организации и типы ВС	95
8.3.	Параллельная обработка информации: уровни и способы организации	96
8.4.	Реализация в многомашиных и многопроцессорных ВС.....	97
8.5.	Операционные контейнеры	97
8.6.	Векторные, матричные, ассоциативные системы.....	98
8.7.	Однородные системы и среды: RISC-архитектуры	99
8.8.	Развитие архитектур, ориентированных на языковые средства и среду программирования.....	99
8.9.	Основы метрической теории ВС.....	100
8.10.	Технология распределенной обработки данных	102
9.	Задачи сортировки Анализ сложности и эффективности алгоритмов сортировки	104
9.1.	Задачи сортировки.....	104
9.2.	Внутренняя и внешняя сортировки	104
9.3.	Алгоритмы сортировки	104
9.4.	Анализ сложности и эффективности алгоритмов поиска и сортировки.....	106
10.	Современные технологии разработки программного обеспечения Управление версиями Документирование.....	108
10.1.	Современные технологии разработки программного обеспечения, постановка задачи, оценка осуществимости.....	108
10.2.	Планирование, тестирование, обеспечение оценки качества.....	110
10.3.	Групповая разработка, управление версиями, организация коллектива разработчиков, документирование.....	111
10.4.	Структурное проектирование, CASE-средства, реинжиниринг программных систем..	115

МАТЕРИАЛ ДЛЯ ПОДГОТОВКИ

Задачи поиска: исчерпывающий поиск, быстрый поиск, использование деревьев в задачах поиска

Задачи поиска

Задача данного действия заключается в нахождении одного или нескольких элементов в множестве, причем искомые элементы должны обладать определенным свойством. Это свойство может быть абсолютным или относительным. Относительное свойство характеризует элемент по отношению к другим элементам: например, минимальный элемент в множестве чисел. Абсолютное свойство - поиск элемента, ключ которого равен заданному «аргументу поиска» x .

Таким образом, в задаче поиска имеются следующие шаги:

- 1) вычисление свойства элемента;
- 2) сравнение свойства элемента с эталонным свойством (для абсолютных свойств) или сравнение свойств двух элементов (для относительных свойств);
- 3) перебор элементов множества.

Исчерпывающий поиск: перебор с возвратом, метод ветвей и границ, динамическое программирование

Исчерпывающий поиск – это процесс нахождения в некотором множестве всех возможных вариантов, среди которых имеется решение конкретной задачи.

Существуют два общих метода организации исчерпывающего поиска: **перебор с возвратом** (*backtracking*) и его естественное логическое дополнение - **метод решета**.

Основная идея поиска с возвратом: построение решения по одному компоненту и выяснение, может ли дальнейшее построение привести к решению. Если да – решение продолжается путем выбора первого допустимого варианта для следующего компонента. Если таких вариантов нет, то такие варианты для всех оставшихся компонентов не рассматриваются. Алгоритм в этой ситуации возвращается к последнему построенному компоненту и заменяет его следующим допустимым компонентом этого уровня.

Общий алгоритм перебора с возвратом на псевдокоде:

```
do { выбора варианта (n): if (подходит) { запись варианта ( );  
  if (решение неполное) { try (n+-...)  
    if ( ! удача) стирание варианта ( ); } }  
while ( !удача || нет вариантов);
```

Способы сокращения перебора

1. Отбрасывание заведомо неверных вариантов.
2. Применение эвристик.
3. Метод ветвей и границ.

Эвристика в программировании – это решение задачи при помощи разбора частных случаев, так как каждая ветвь эвристического алгоритма эффективнее, чем общее решение. Таким образом эвристический алгоритм не решает задачу в общем виде, а работает только в определенных случаях, но это и хорошо, так как некоторые задачи не имеют общего решения и они решаются только при помощи эвристики. Применяется тогда, когда мы ищем одно из возможных решений, а не все. Таким образом эвристика позволяет выбрать из всех вариантов самый вероятный.

Перебор с возвратом (backtracking) – это один из методов организации исчерпывающего поиска, который строится конструктивно последовательным расширением частичного решения.

Метод решета – это один из методов организации исчерпывающего поиска, при котором из множества возможных вариантов исключаются все элементы, не являющиеся решениями.

Незначительные модификации метода перебора с возвратом, связанные с представлением данных или особенностями реализации, имеют и иные названия: метод ветвей и границ, поиск в глубину, метод проб и ошибок и т. д.

Процедура ветвления состоит в разбиении области допустимых решений на подобласти меньших размеров. Процедуру можно рекурсивно применять к подобластям. Полученные подобласти образуют дерево, называемое деревом поиска или деревом ветвей и границ. Узлами этого дерева являются построенные подобласти.

Процедура нахождения оценок заключается в поиске верхних и нижних границ для оптимального значения на подобласти допустимых решений.

Метод ветвей и границ — общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и комбинаторной оптимизации. По существу, метод является вариацией полного перебора с отсеком подмножеств допустимых решений, заведомо не содержащих оптимальных решений.

Метод ветвей и границ был впервые предложен в 1960 году Ленд и Дойг для решения задач целочисленного программирования.

В основе метода ветвей и границ лежит следующая идея (*для задачи минимизации*): если нижняя граница для подобласти А дерева поиска больше, чем верхняя граница какой-либо ранее рассмотренной подобласти В, то А может быть исключена из дальнейшего рассмотрения (правило отсева). Обычно, минимальную из полученных верхних оценок записывают в глобальную переменную m ; любой узел дерева поиска, нижняя граница которого больше значения m , может быть исключен из дальнейшего рассмотрения.

Если нижняя граница для узла дерева совпадает с верхней границей, то это значение является минимумом функции и достигается на соответствующей подобласти.

Общая идея метода может быть описана на примере поиска минимума и максимума функции $f(x)$ на множестве допустимых значений x . Функция f от x может быть произвольной природы. Для метода ветвей и границ необходимы две процедуры: ветвление и нахождение оценок (границ).

В отличие от метода перебора, который ищет допустимое решение конкретной задачи, метод ветвей и границ ищет оптимальное решение, т.е. допустимое решение с наилучшим значением целевой функции.

Метод ветвей и границ требует:

1) Способ получения для каждого узла дерева границ наилучшего значения функции для всех решений (эта граница должна быть нижней для задачи минимизации и верхней для задачи максимизации).

2) Значения наилучшего решения, полученного к этому моменту.

Если такая информация доступна, то можно сравнивать значения границ узла со значением полученного к этому моменту решения. Если значения границ не лучше значения уже имеющегося наилучшего решения, то такой узел является бесперспективным и его обработка может быть завершена.

Динамическое программирование

Часто, в случае большого числа повторений решения одной и той же подзадачи, можно получать эффективные алгоритмы с помощью табличной техники, называемой *динамическим программированием*. Динамическое программирование предполагает последовательное решение всех подзадач исходной задачи в порядке возрастания их размера с запоминанием ответов в специальной таблице по мере их получения. Преимущество этого метода по сравнению с рекурсивным алгоритмом состоит в том, что раз уж подзадача решена, ее ответ размещен в таблицу и в дальнейшем никогда не вычисляется повторно. Ответ подзадачи берется из таблицы каждый раз, когда он нужен для нахождения решения подзадачи большего размера. Понятно, что эта техника позволяет получать эффективные алгоритмы в тех случаях, когда объем хранимой в таблице информации оказывается не слишком большим.

Динамическое программирование обычно придерживается двух подходов к решению задач:

нисходящее динамическое программирование: задача разбивается на подзадачи меньшего размера, они решаются и затем комбинируются для решения исходной задачи. Используется запоминание для решений уже решенных подзадач.

восходящее динамическое программирование: все подзадачи, которые впоследствии понадобятся для решения исходной задачи просчитываются заранее и затем используются для

построения решения исходной задачи. Этот способ лучше нисходящего программирования в смысле размера необходимого стека и количества вызова функций, но иногда бывает нелегко заранее выяснить, решение каких подзадач нам потребуется в дальнейшем.

Быстрый поиск: бинарный и последовательный поиски в массивах, хеширование

Под поиском в массиве будем понимать задачу нахождения индекса, по которому в массиве располагается некоторый заданный элемент.

Если заранее известна некоторая информация о данных, среди которых ведется поиск, например, известно, что массив данных отсортирован, то удастся сократить время поиска, используя **бинарный поиск**.

Итак, требуется определить место X в отсортированном (например, в порядке возрастания) массиве A . Делим пополам и сравниваем X с элементом, который находится на границе этих половин. Отсортированность A позволяет по результату сравнения со средним элементом массива исключить из рассмотрения одну из половин.

Каждое сравнение уменьшает диапазон поиска приблизительно в два раза. Следовательно, общее количество сравнений имеет порядок $O(\log N)$.

Последовательный поиск – тривиальный алгоритм поиска, заключается в последовательном переборе элементов массива до тех пор, пока не будет обнаружен искомый или не будут просмотрены все элементы массива. В случае, когда нет никакой дополнительной информации о характере расположения элементов в массиве, такой алгоритм представляется единственно возможным. Нетрудно подсчитать, что, в худшем случае, для поиска в массиве, состоящем из N элементов, потребуется N операций сравнения.

Хеширование представляет собой преобразование любого объема информации в уникальный набор символов, который присущ только этому массиву входящей информации. Этот набор символов и будет называться хэшем.

У хэш-функции есть несколько обязательных свойств:

- Хэш всегда уникален для каждого массива информации. Однако иногда случаются так называемые коллизии, когда для разных входных блоков информации вычисляются одинаковые хэш-коды.
- При самом незначительном изменении входной информации ее хэш полностью меняется.
- Хэш-функция необратима и не позволяет восстанавливать исходный массив информации из символьной строки. Это можно сделать, только перебрав все возможные варианты, что при бесконечном количестве информации требует много времени и денег.
- Хеширование позволяет достаточно быстро вычислить нужный хэш для достаточно большого объема информации.
- Алгоритм работы хэш-функции, как правило, делается открытым, чтобы при необходимости можно было оценить ее стойкость к восстановлению начальных данных по выдаваемому хэшу.
- Хэш-функция должна уметь приводить любой объем данных к числу заданной длины.

Использование деревьев в задачах поиска: бинарные и случайные бинарные, оптимальные и сбалансированные деревья поиска

Случайные бинарные деревья поиска

Случайные деревья поиска представляют собой упорядоченные бинарные деревья поиска, при создании которых элементы (их ключи) вставляются в случайном порядке.

При создании таких деревьев используется тот же алгоритм, что и при добавлении вершины в бинарное дерево поиска. Будет ли созданное дерево случайным или нет, зависит от того, в каком порядке поступают элементы для добавления. Примеры различных деревьев, создаваемых при различном порядке поступления элементов приведены ниже.

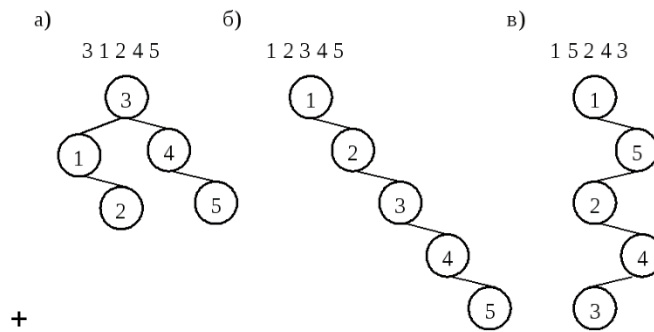


Рисунок 3.15 – Случайные и вырожденные деревья поиска

При поступлении элементов в случайном порядке получаем дерево с минимальной высотой h (см. рис. 3.15.а), а соответственно минимизируется время поиска элемента в таком дереве, которое пропорционально $O(\log n)$. При поступлении элементов в упорядоченном виде (см. рис. 3.12.б) или в несколько необычном порядке (см. рис. 3.12.в) происходит построение вырожденных деревьев поиска (оно вырождено в линейный список), что несколько не сокращает время поиска, которое составляет $O(n)$.

Бинарные (двоичные) деревья

Для организации поиска в основной памяти особое значение имеют упорядоченные двоичные (бинарные) деревья (как, например, на рисунке 10.3). В каждом таком дереве естественно определяются левое и правое поддеревья. Двоичное дерево называется **идеально сбалансированным**, если число вершин в его левом и правом поддеревьях отличается не более, чем на 1 (легко видеть, что при соблюдении этого условия длины пути до любой листовой вершины дерева отличаются не больше, чем на 1). Примеры идеально сбалансированных деревьев показаны на рисунке 10.5.

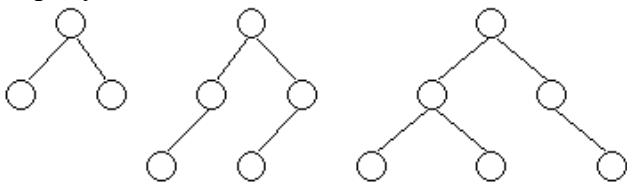


Рис. 10.5.

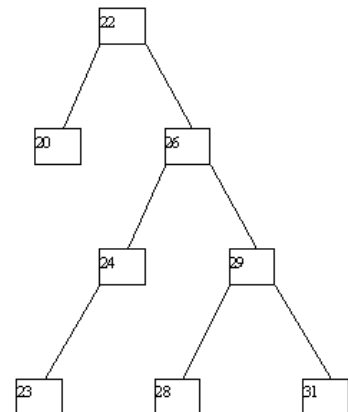


Рис. 10.6.

Двоичные деревья обычно представляются как динамические структуры с базовым типом записи T , в число полей которого входят два указателя на переменные типа T .

При использовании в целях поиска элементов данных по значению уникального ключа применяются двоичные деревья поиска, обладающие тем свойством, что для любой вершины дерева значение ее ключа больше значения ключа любой вершины ее левого поддерева и больше значения ключа любой вершины правого поддерева (рисунок 10.6). Для поиска заданного ключа в дереве поиска достаточно пройти по одному пути от корня до (возможно, листовой) вершины (рисунок 10.7). Высота идеально сбалансированного двоичного дерева с n вершинами составляет не более, чем $\log n$ (логарифм двоичный), поэтому при применении таких деревьев в качестве деревьев поиска (рисунок 8) потребуется не более $\log n$ сравнений.

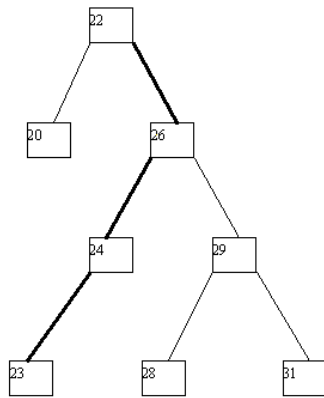


Рис.10.7. Путь поиска ключа по значению "23"

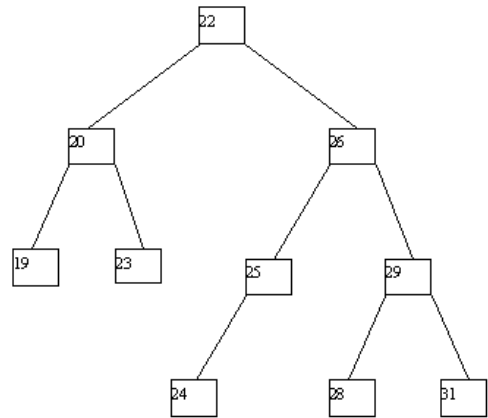


Рис. 10.8. Идеально сбалансированное двоичное дерево

Сбалансированные двоичные деревья поиска

Идеально сбалансированные деревья представляют, в большей степени, чисто теоретический интерес, поскольку поддержание идеальной сбалансированности требует слишком больших накладных расходов. В 1962 г. советские математики Адельсон-Вельский и Ландис предложили менее строгое определение сбалансированности деревьев, которое в достаточной степени обеспечивает возможности использования сбалансированных деревьев при существенно меньших расходах на поддержание сбалансированности.

Такие деревья принято называть АВЛ-деревьями (в соответствии с именами их первооткрывателей).

По определению, двоичное дерево называется сбалансированным (или АВЛ) деревом в том и только в том случае, когда высоты двух поддеревьев каждой из вершин дерева отличаются не более, чем на единицу. При использовании деревьев, соответствующих этому определению, обеспечивается простая процедура балансировки при том, что средняя длина поиска составляет $O(\log n)$, т.е. практически не отличается от длины поиска в идеально сбалансированных деревьях.

Известно, что оценкой стоимости поиска в АВЛ-дереве, а также выполнения операций включения и исключения ключей является $O(\log n)$, т.е. эти деревья при поиске ведут себя почти так же хорошо, как и идеально сбалансированные деревья, а поддержка балансировки при включениях и исключениях обходится гораздо дешевле.

Деревья оптимального поиска

При поиске в двоичном дереве одни элементы могут искаться чаще, чем другие, то есть существуют вероятности P_k поиска k -го элемента и для различных элементов эти вероятности неодинаковы. Можно сразу предположить, что поиск в дереве в среднем будет более быстрым, если те элементы, которые ищутся чаще, будут находиться ближе к корню дерева.

Дерево поиска называется оптимальным, если его цена минимальна или, другими словами, **оптимальное бинарное дерево поиска** – это бинарное дерево поиска, построенное в расчете на обеспечение максимальной производительности при заданном распределении вероятностей поиска требуемых данных.

Существует подход построения оптимальных деревьев поиска, при котором элементы вставляются в порядке уменьшения частот, что дает в среднем неплохие деревья поиска. Однако этот подход может дать вырожденное дерево поиска, которое будет далеко от оптимального.

Еще один подход состоит в выборе корня k таким образом, чтобы максимальная сумма вероятностей для вершин левого поддерева или правого поддерева была настолько мала, насколько это возможно. Такой подход также может оказаться плохим в случае выбора в качестве корня элемента с малым значением p_k .

Процедура построения дерева оптимального поиска достаточно сложна и опирается на тот факт, что любое поддерево дерева оптимального поиска также обладает свойством оптимальности. Поэтому известный алгоритм строит дерево "снизу-вверх", т.е. от листьев к корню. Сложность этого алгоритма и расходы по памяти составляют $O(n^2)$. Имеется эвристический алгоритм, дающий дерево, близкое к оптимальному, со сложностью $O(n \cdot \log n)$ и расходами памяти - $O(n)$.

Таким образом, создание оптимальных деревьев поиска требует больших накладных затрат, что не всегда оправдывает выигрыш при быстром поиске.

Уровни моделей и этапы проектирования баз данных Инфологическое моделирование

Базы данных (БД) и системы управления базой данных (СУБД)

Ба́за да́нных — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных.

В классификацию *по модели данных* обычно включают:

- иерархические;
- объектные или объектно-ориентированные;
- объектно-реляционные;
- реляционные;
- сетевые;
- функциональные.

Классификация *по среде хранения* различает базы данных, хранящие данные во вторичной памяти («традиционные»), *резидентные* (все данные на стадии исполнения находятся в оперативной памяти) и *третичные*, хранящие данные на отсоединяемых устройствах массового хранения — на основе магнитных лент или оптических дисков. При этом во всех классах так или иначе используются все среды хранения, например, для резидентных баз данных СУБД записывает в постоянную память журналы предзаписи, а для традиционных баз используется кэш в оперативной памяти.

Также базы данных могут классифицироваться *по содержанию*, например, могут быть географическими, историческими, научными, мультимедийными. Для некоторых форм содержания строятся специализированные СУБД, либо добавляются специализированные возможности в СУБД общего назначения, среди таковых баз данных:

- пространственные: базы с пространственными свойствами сущностей предметной области, используются в геоинформационных системах;
- временные (темпоральные): поддерживают какой-либо *аспект времени*, не считая времени, определяемого пользователем.

По степени распределённости базы данных подразделяются на централизованные (сосредоточенные) — полностью поддерживаемые на одном оборудовании, и распределённые. Среди многообразия вариантов распределённых баз данных выделяются:

- сегментированные: разделённые на части под управлением различных экземпляров СУБД по какому-либо критерию;
- тиражированные (реплицированные; англ. *replicated database*): одни и те же данные разнесены под управление различных экземпляров СУБД;
- неоднородные (англ. *heterogeneous distributed database*): фрагменты распределённой базы в разных узлах сети поддерживаются средствами более одной СУБД.

Возможны смешанные варианты, например, для одной и той же распределённой базы для больших объектов используется сегментирование, а для небольших — репликация.

Система управления базами данных, сокр. СУБД (Database Management System) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

СУБД — комплекс программ, позволяющих создать базу данных (БД) и манипулировать данными (вставлять, обновлять, удалять и выбирать). Система обеспечивает безопасность, надёжность хранения и целостность данных, а также предоставляет средства для администрирования БД.

Основные функции СУБД:

- управление данными во внешней памяти (на дисках);
- управление данными в оперативной памяти с использованием дискового кэша;
- журнализация изменений, резервное копирование и восстановление базы данных после сбоев;
- поддержка языков БД (язык определения данных, язык манипулирования данными).

Состав СУБД:

Обычно современная СУБД содержит следующие компоненты:

- **ядро**, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию;
- **процессор языка базы данных**, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода;
- **подсистему поддержки времени исполнения**, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД;
- **сервисные программы** (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы.

Классификация СУБД:

По модели данных

Примеры:

- Иерархические
- Сетевые
- Реляционные
- Объектно-ориентированные
- Объектно-реляционные

По степени распределённости

- Локальные СУБД (все части локальной СУБД размещаются на одном компьютере)
- Распределённые СУБД (части СУБД могут размещаться не только на одном, но на двух и более компьютерах).

По способу доступа к БД

- *Файл-серверные*

В файл-серверных СУБД файлы данных располагаются централизованно на файл-сервере. СУБД располагается на каждом клиентском компьютере (рабочей станции). Доступ СУБД к данным осуществляется через локальную сеть. Синхронизация чтений и обновлений осуществляется посредством файловых блокировок.

Преимуществом этой архитектуры является низкая нагрузка на процессор файлового сервера.

Недостатки: потенциально высокая загрузка локальной сети; затруднённость или невозможность централизованного управления; затруднённость или невозможность обеспечения таких важных характеристик, как высокая надёжность, высокая доступность и высокая безопасность. Применяются чаще всего в локальных приложениях, которые используют функции управления БД; в системах с низкой интенсивностью обработки данных и низкими пиковыми нагрузками на БД.

На данный момент файл-серверная технология считается устаревшей, а её использование в крупных информационных системах — недостатком^[3].

Примеры: Microsoft Access, Paradox, dBase, FoxPro, Visual FoxPro.

- *Клиент-серверные*

Клиент-серверная СУБД располагается на сервере вместе с БД и осуществляет доступ к БД непосредственно, в монопольном режиме. Все клиентские запросы на обработку данных обрабатываются клиент-серверной СУБД централизованно.

Недостаток клиент-серверных СУБД состоит в повышенных требованиях к серверу.

Достоинства: потенциально более низкая загрузка локальной сети; удобство централизованного управления; удобство обеспечения таких важных характеристик, как высокая надёжность, высокая доступность и высокая безопасность.

Примеры: Oracle Database, Firebird, Interbase, IBM DB2, Informix, MS SQL Server, Sybase Adaptive Server Enterprise, PostgreSQL, MySQL, Caché, ЛИНТЕР.

- *Встраиваемые*

Встраиваемая СУБД — СУБД, которая может поставляться как составная часть некоторого программного продукта, не требуя процедуры самостоятельной установки. Встраиваемая СУБД предназначена для локального хранения данных своего приложения и не рассчитана на коллективное использование в сети.

Физически встраиваемая СУБД чаще всего реализована в виде подключаемой библиотеки. Доступ к данным со стороны приложения может происходить через SQL либо через специальные программные интерфейсы.

Примеры: OpenEdge, SQLite, BerkeleyDB, Firebird Embedded, Microsoft SQL Server Compact, ЛИНТЕР.

Выбор системы управления базами данных

Система управления базами данных (далее СУБД) - система управления базами данных (БД) (DBMS) отвечает за агрегирование данных и их последующее хранение и обработку.

Реляционная модель - представление БД в виде таблиц для действий над записями на языке SQL.

«Постреляционная СУБД» - наличие в реляционной СУБД файлов управления данными, не вписывающихся в реляционную модель, т.е. объектов. Ранее данные хранились только в алфавитно-цифровой форме, классифицировались по стандартным типам (строки, целые числа и т.д.). Теперь сюда включаются и бинарные объекты: изображения, видео и большие фрагменты текста, по которым может происходить поиск.

Различают реляционные базы данных, постреляционные и NoSQL, но еще существуют и их разновидности:

№	Тип СУБД	Когда выбирать	Популярные СУБД данного типа
1	Реляционные	Нужна транзакционность; высокая нормализация; большая доля операций на вставку	Oracle, MySQL, Microsoft SQL Server, PostgreSQL, IBM DB2, SQLite
2	Ключ-значение	Задачи кэширования и брокеры сообщений	Redis, Memcached, etcd
3	Документные	Для хранения объектов в одной сущности, но с разной структурой; хранение структур на основе JSON	Couchbase, MongoDB, Amazon DocumentDB
4	Графовые	Задачи подобные социальным сетям; системы оценок и рекомендаций	Neo4j, Amazon Neptune, InfiniteGraph, TigerGraph
5	Колоночные	Хранилища данных; выборки со сложными аналитическими вычислениями; количество строк в таблице превышает сотни миллионов	Vertica, ClickHouse, Google BigQuery, Sybase \ SAP IQ, InfoBright
6	Time series	Системы мониторинга, сбора телеметрии, и финансовые системы, с привязкой к временным меткам или временным рядам	InfluxDB, Kdb+, Prometheus, TimescaleDB, QuestDB, AWS Timestream, OpenTSDB, GridDB
7	Объектные	Высокопроизводительная обработка данных, имеющих сложную структуру, с использованием языков объектно ориентированного программирования	MongoDB Realm, InterSystems Caché, ObjectStore, Actian NoSQL DB, Objectivity/DB
8	Search engine	Системы полнотекстового поиска	Apache Solr, Elasticsearch, Splunk
9	Spatial	GIS-решения, работа с геометрическими объектами	Oracle Spatial, Microsoft SQL, PostGIS, SpatialLite

При выборе типа СУБД следует, прежде всего, исходить из типа решаемых задач, типов обрабатываемых данных, перспектив роста и масштабирования, на популярность и наличие широкого круга разработчиков и средств разработки.

Жизненный цикл базы данных

Жизненный цикл базы данных (ЖЦ БД) – это процедура создания концептуальной схемы базы данных, определения данных, включаемых в базу данных создание программ обновления и обработки данных.

ЖЦ БД – это процесс проектирования, реализации и поддержания системы базы данных.

Жизненный цикл базы данных состоит из шести этапов:

1. Предварительное планирование
2. Проверка осуществимости
3. Определение требований

4. Концептуальное проектирование
5. Реализация
6. Оценка работы и поддержка базы данных.

Предварительное планирование конкретной системы управления базами данных осуществляется в процессе разработки стратегического плана. Ищутся ответы на следующие вопросы:

1. Сколько используется прикладных программ и какие функции они выполняют?
2. Какие файлы связаны с каждым из этих приложений?
3. Какие новые приложения и файлы находятся в процессе создания?

Собранная информация может быть использована для того, чтобы установить связи между текущими приложениями и определить, как используется информация приложений. Она также помогает определить будущие требования к системе и получить от системы базы данных экономическую выгоду. Информация документируется в виде обобщенной концептуальной модели данных.

Проверка осуществимости включает подготовку отчетов по следующим вопросам:

1. Технологическая осуществимость. Есть ли технология, необходимая для реализации запланированной базы данных?
2. Операционная осуществимость. Располагает ли компания персоналом, средствами и экспертами, необходимыми для успешного осуществления плана создания базы данных?
3. Экономическая целесообразность. Можно ли определить выгоды? Окупается ли запланированная система? Можно ли измерить издержки и выгоду?

Определение требований включает выбор целей базы данных, выяснение информационных потребностей различных отделов и руководителей компании и требований к оборудованию и программному обеспечению.

Этап *концептуального проектирования* включает создание концептуальной схемы базы данных. Спецификации разрабатываются в той степени, которая требуется для перехода к реализации. На этом этапе создаются подробные модели пользовательских представлений данных; затем они интегрируются в концептуальную модель, фиксирующую все элементы корпоративных данных, которые будет содержать база данных.

В *процессе реализации базы данных* выбирается и приобретается СУБД, затем подробная концептуальная модель превращается в проект реализации базы данных; создается словарь данных, база данных наполняется данными, создаются прикладные программы и обучается пользователь.

Оценка включает опрос пользователей с целью выяснения, какие информационные потребности остались неучтенными. В случае необходимости вносятся изменения. Обеспечивается поддержка системы путем внесения изменений и добавления новых программ и элементов данных по мере расширения и изменения потребностей бизнеса.

Уровни моделей и этапы проектирования БД

В процессе научных исследований, посвященных тому, как именно должна быть устроена СУБД, предлагались различные способы реализации. Самым жизнеспособным из них оказалась предложенная американским комитетом по стандартизации ANSI (American National Standards Institute) трехуровневая система организации БД, изображенная на рис. 2.1:

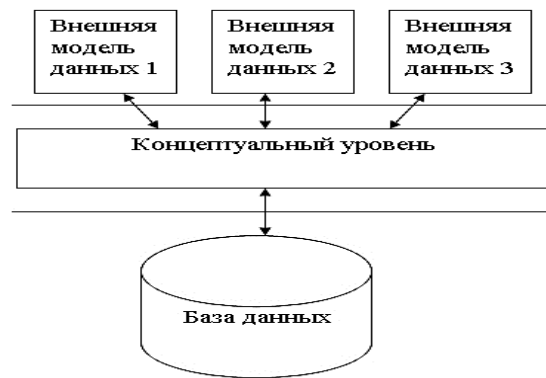


Рис. 2.1. Трехуровневая модель системы управления базой данных, предложенная ANSI

1. **Уровень внешних моделей** - самый верхний уровень, где каждая модель имеет свое «видение» данных. Этот уровень определяет точку зрения на БД отдельных приложений. Каждое приложение видит и обрабатывает только те данные, которые необходимы именно этому приложению. Например, система распределения работ использует сведения о квалификации сотрудника, но ее не интересуют сведения об окладе, домашнем адресе и телефоне сотрудника, и наоборот, именно эти сведения используются в подсистеме отдела кадров.

2. **Концептуальный уровень** - центральное управляющее звено, здесь база данных представлена в наиболее общем виде, который объединяет данные, используемые всеми приложениями, работающими с данной базой данных. Фактически концептуальный уровень отражает обобщенную модель предметной области (объектов реального мира), для которой создавалась база данных. Как любая модель, концептуальная модель отражает только существенные, с точки зрения обработки, особенности объектов реального мира.

3. **Физический уровень** - собственно данные, расположенные в файлах или в страничных структурах, расположенных на внешних носителях информации.

Эта архитектура позволяет обеспечить логическую (между уровнями 1 и 2) и физическую (между уровнями 2 и 3) независимость при работе с данными. Логическая независимость предполагает возможность изменения одного приложения без корректировки других приложений, работающих с этой же базой данных. Физическая независимость предполагает возможность переноса хранимой информации с одних носителей на другие при сохранении работоспособности всех приложений, работающих с данной базой данных. Это именно то, чего не хватало при использовании файловых систем.

Выделение концептуального уровня позволило разработать аппарат централизованного управления базой данных.



Рис. 2.3. Классификация моделей данных

В соответствии с рассмотренной ранее трехуровневой архитектурой мы сталкиваемся с понятием модели данных **по отношению к каждому уровню**. И действительно, **физическая модель данных** оперирует категориями, касающимися организации внешней памяти и структур

хранения, используемых в данной операционной среде. В настоящий момент в качестве физических моделей используются различные методы размещения данных, основанные на файловых структурах: это организация файлов прямого и последовательного доступа, индексных файлов и инвертированных файлов, файлов, использующих различные методы хэширования, взаимосвязанных файлов. Кроме того, современные СУБД широко используют страничную организацию данных. *Физические модели данных*, основанные на страничной организации, являются наиболее перспективными.

Наибольший интерес вызывают модели данных, используемые **на концептуальном уровне**. По отношению к ним внешние модели называются подсхемами и используют те же абстрактные категории, что и концептуальные модели данных.

Кроме трех рассмотренных уровней абстракции при проектировании БД существует еще один уровень, предшествующий им. Модель этого уровня должна выражать информацию о предметной области в виде, независимом от используемой СУБД. Эти модели называются инфологическими, или семантическими, и отражают в естественной и удобной для разработчиков и других пользователей форме информационно-логический уровень абстрагирования, связанный с фиксацией и описанием объектов предметной области, их свойств и их взаимосвязей.

Инфологические модели данных используются на ранних стадиях проектирования для описания структур данных в процессе разработки приложения, а *дата-логические* модели уже поддерживаются конкретной СУБД.

Этапы проектирования баз данных

Реализацию сложных проектов по созданию информационных систем (ИС) принято разбивать на стадии анализа, проектирования, кодирования, тестирования и сопровождения. Известно, что исправление ошибок, допущенных на предыдущей стадии, обходится примерно в 10 раз дороже, чем на текущей, следовательно, наиболее критическими являются первые стадии проекта – анализа и проектирования.

Процесс разработки БД можно разбить на несколько этапов:

- Исследование предметной области;
- Инфологическое моделирование;
- Даталогическое проектирование;
- Даталогическое моделирование;
- Проектирование на физическом уровне.

Инфологическое моделирование

Данный этап заключается в создании концептуальной модели данных для анализируемой части предприятия. Модель данных создается на основе информации, записанной в спецификациях требований пользователей. Концептуальное проектирование базы данных абсолютно не зависит от таких подробностей ее реализации, как тип выбранной целевой СУБД, набор создаваемых прикладных программ, используемые языки программирования, тип выбранной вычислительной платформы, а также от любых других особенностей физической реализации. При разработке концептуальная модель данных постоянно подвергается тестированию и проверке на соответствие требованиям пользователей. Созданная концептуальная модель данных предприятия является источником информации для этапа логического проектирования базы данных.

Концептуальная модель строится с использованием стандартных языковых средств, обычно графических, например ER-диаграмм (диаграмм «Сущность-связь»). Такая модель строится без ориентации на какую-либо конкретную СУБД.

Основные элементы данной модели:

1. Описание объектов предметной области и связей между ними.
2. Описание информационных потребностей пользователей (описание основных запросов к БД).
3. Описание алгоритмических зависимостей между данными.

4. Описание ограничений целостности, т.е. требований к допустимым значениям данных и к связям между ними.

Языковые средства современных СУБД

Функциональные возможности поддерживаемой средствами СУБД модели данных становятся доступными пользователю благодаря ее языковым средствам. Языковые средства СУБД используются для выполнения двух основных функций - для описания представления базы данных на управляемых уровнях системной архитектуры и для выполнения операций манипулирования данными.

Язык описания данных (ЯОД) предназначен для задания схемы базы данных, которая включает описание структуры базы данных и налагаемых на нее ограничений целостности в рамках правил, регламентированных той моделью данных, которая поддерживается рассматриваемой СУБД. Помимо указанных функций, ЯОД обеспечивают также возможности задания ограничений доступа к данным или полномочий пользователей.

Язык манипулирования данными (ЯМД) позволяет запрашивать предусмотренные в системе операции над данными из базы данных. После выполнения оператора, записанного на ЯМД, информационное заполнение базы данных изменяется

Язык запросов (ЯЗ) позволяет выбирать данные из БД, агрегировать и подвергать всевозможной аналитической обработке.

Аналогично языку определения данных ЯМД не обязательно выступает в форме синтаксически самостоятельного языка СУБД. На практике разделение ЯОД и ЯМД играет скорее методическую роль или используется в технологических целях.

ЯОД, ЯМД и ЯЗ не всегда синтаксически оформляются в виде самостоятельных языков. Наоборот, в настоящее время все они вошли в состав единого реляционного языка SQL. С 1986 г. был принят ряд версий международного стандарта SQL. Он используется в большинстве коммерческих реляционных СУБД, в том числе и на персональных компьютерах.

Некоторые СУБД располагают такими языками, которые не только реализуют функции определения данных и манипулирования данными, но и обладают средствами, свойственными универсальным языкам программирования. Благодаря этому они могут использоваться как функционально полное инструментальное средство для создания приложений систем баз данных. В качестве примера приведем языки систем dBase, Clipper, Paradox.

Для того чтобы иметь развитые средства разработки приложений, в СУБД обеспечиваются интерфейсы прикладного программирования. Приложения для таких систем могут разрабатываться с помощью расширения традиционного языка программирования операторами (командами, функциями, процедурами и т.п.) указанного интерфейса. Благодаря этому будет восполняться функциональная неполнота языков данной системы. Язык программирования выступает при этом в роли включающего языка по отношению к языку интерфейса прикладного программирования СУБД, и прикладные системы реализуются на таком расширенном языке. Интерфейсы прикладного программирования предусмотрены во многих СУБД.

В Web-программировании активно используется СУБД MySQL. Для работы с БД этой системы применяют язык программирования PHP. Это Си-подобный язык, предназначенный для быстрого создания программ на Web-сервере.

Даталогическое моделирование

Его цель состоит в создании логической модели данных для исследуемой части предприятия. Концептуальная модель данных, созданная на предыдущем этапе, уточняется и преобразуется в логическую модель данных. Логическая модель данных учитывает особенности выбранной модели организации данных в целевой СУБД (например, реляционная модель).

Если концептуальная модель данных не зависит от любых физических аспектов реализации, то логическая модель данных создается на основе выбранной модели организации данных целевой СУБД. Созданная логическая модель данных является источником информации для

этапа физического проектирования и обеспечивает разработчика физической базы данных средствами поиска компромиссов, необходимых для достижения поставленных целей, что очень важно для эффективного проектирования.

При правильно организованном сопровождении поддерживаемая в актуальном состоянии модель данных позволяет точно и наглядно представить любые вносимые в базу данных изменения, а также оценить их влияние на прикладные программы и использование данных, уже имеющихся в базе.

Логическое (дatalogическое) проектирование – отображение инфологической модели на модель данных, используемую в конкретной СУБД, например на реляционную модель данных. Для реляционных СУБД даталогическая модель – набор таблиц, обычно с указанием ключевых полей, связей между таблицами. Если инфологическая модель построена в виде ER-диаграмм (или других формализованных средств), то даталогическое проектирование представляет собой построение таблиц по определённым формализованным правилам, а также нормализацию этих таблиц. Этот этап может быть в значительной степени автоматизирован.

Проектирование на физическом уровне

Физическое проектирование представляет собой организованный процесс подготовки описания реализации базы данных на вторичных запоминающих устройствах; на этом этапе рассматриваются основные отношения, организация файлов и индексов, предназначенных для обеспечения эффективного доступа к данным, а также все связанные с этим ограничения целостности и средства защиты.

Физическое проектирование является третьим и последним этапом создания проекта базы данных, при выполнении которого проектировщик принимает решения о способах реализации разрабатываемой базы данных. Во время предыдущего этапа проектирования была определена логическая структура базы данных, которая описывает отношения и ограничения в рассматриваемой прикладной области. Хотя эта структура не зависит от конкретной целевой СУБД, она создается с учетом выбранной модели хранения данных, например реляционной, сетевой или иерархической. Однако, приступая к физическому проектированию базы данных, прежде всего необходимо выбрать конкретную целевую СУБД. Поэтому физическое проектирование неразрывно связано с конкретной СУБД. Между логическим и физическим проектированием существует постоянная обратная связь, так как решения, принимаемые на этапе физического проектирования с целью повышения производительности системы, способны повлиять на структуру логической модели данных.

Средства и методы проектирование БД

В теории баз данных существует ряд методов разработки моделей БД, отображающих разные уровни её архитектуры. Распространены два основных подхода к проектированию систем баз данных: "нисходящий" и "восходящий":

При «восходящем» подходе осуществляют структурное проектирование снизу—вверх. Этот процесс называют синтезом, попыткой получения целого (адекватно отображающего описание предметной области) на основе описания составляющих его частей.

«Восходящее» проектирование – это достаточно сложная и **устаревшая** методика, которая подходит для проектирования только небольших баз данных.

При «нисходящем» проектировании осуществляется структурное проектирование сверху—вниз. Такой процесс называют анализом – происходит изучение целого (описания предметной области), затем разделение целого на составные части и далее следует последовательное изучение этих частей.

Метод «нисходящего» проектирования достаточно формализован и используется в *CASE* (*Computer Aided System /Software Engineering* — компьютерное проектирование программного обеспечения и систем) средствах. Проведение тщательного анализа предметной области, выявление всех присущих ей классов объектов и связей между ними, правильное их отображение в ИЛМ предметной области, ведет к получению высоко нормализованной схемы логической структуры

реляционной БД.

Проектирование БД может быть автоматизировано. Для этого используются различные CASE – средства, особенно эффективно их использование при создании крупных корпоративных АИС большим коллективом разработчиков. Использование современных CASE – средств позволяет поддерживать как начальные этапы разработки АИС, так и проектирование, и генерацию баз данных и пользовательских интерфейсов. CASE – средства обеспечивают качество принимаемых технических решений и подготовку проектной документации.

CASE—средства классифицируются по методологиям проектирования (структурно—ориентированные, объектно—ориентированные, комплексно—ориентированные), по графическим нотациям построения диаграмм, по степени интегрированности (отдельные локальные средства, набор интегрированных средств, охватывающих большинство этапов разработки АИС), по режиму коллективной разработки проекта (режим реального времени, режим объединения проектов) и ряду других. Современный рынок программных средств насчитывает около 300 различных CASE – средств

Ограничения целостности

Целостностью данных можно назвать механизм поддержания соответствия базы данных предметной области.

В реляционной модели данных определяют две базовые категории обеспечения целостности:

Целостность ссылок.

Целостность сущностей.

Под целостностью понимают правильность данных в любой момент времени.

Данной цели можно достигнуть только в определенных пределах: СУБД не может выполнять контроль правильности каждого отдельного значения, которое вводится в базу данных (несмотря на то, что можно выполнить проверку каждого значения на правдоподобность). К примеру, невозможно проверить, что введенное значение 7, которое представляет номер дня недели, *на самом деле* должно быть равным 4. Но значение 8 однозначно будет являться ошибочным и база данных должна отвергнуть такое значение. В таком случае СУБД необходимо сообщить, что номера дней недели должны выбираться из набора чисел (от 1 до 7).

Поддержание целостности БД может быть рассмотрено как обеспечение защиты данных от разрушений или ошибочных изменений (не стоит путать с незаконными разрушениями и изменениями, которые являются проблемой безопасности).

Ограничения целостности

Ограничения целостности являются специальными средствами баз данных, которые предназначены для предупреждения возможности попадания в базу недопустимым данным (к примеру, не дать пользователю совершить ошибки при введении данных).

Ограничения целостности делятся на 3 основные категории:

1 Средства обеспечения доменной целостности - предназначены для недопущения ввода в поля базы данных недопустимых значений. К примеру, наименование товара должно состоять из букв, а номер телефона - из цифр. Обеспечивается такая целостность в базах данных зачастую установлением условий на значение, ключами, хранимыми процедурами, триггерами, запретом пустых значений.

2 Сущностная целостность, главной задачей которой является недопущение двукратного попадания данных об одной сущности в базу. Такую целостность обеспечивает установление ограничения уникальности и первичный ключ. Она не допускает, чтобы любой атрибут, который принадлежит первичному ключу, мог принимать неопределенное значение.

3 Ссылочную целостность обеспечивает система внешних и первичных ключей. К примеру, с помощью данных средств можно дать гарантию, что в базе не будет заказов, которые оформлены на покупателей, отсутствующих в базе данных.

Внешний ключ должен иметь значение, которое:

Равно значению первичного ключа характеризуемой (ассоциируемой) сущности;

Полностью не определено, то есть каждое значение атрибута, который участвует во внешнем ключе, должно являться неопределённым.

Выделяют еще 2 категории, которые относятся к средствам обеспечения целостности - средства процедурного и декларативного характера.

Средства декларативного характера создают в виде составных частей объектов при их определении в БД (к примеру, установление условия на значение при создании таблицы в БД).

Средства процедурного характера (хранимые процедуры и триггеры) реализованы в виде отдельных программных модулей.

В основном декларативные ограничения являются менее функциональными, но более экономными по отношению к ресурсам и наоборот. Обратим внимание, что использованием развитой системы ограничений целостности определяется зрелость БД. Зачастую легче с самого начала обеспечить непопадание в базу данных неверных значений, чем после убирать их из базы данных.

К тому же, разработчики при создании ограничений целостности должны обеспечить перехват ошибок, которые возникают при нарушениях целостности, клиентским приложением.

Также выделяют целостность, определяемую пользователем. Любая база данных содержит дополнительные специфические правила, относящиеся только к ней и определяющиеся разработчиком. При этом чаще всего контролируют:

Уникальность каких-либо атрибутов,

Диапазон значений (например, оценкой может быть лишь 2,3,4,5);

Принадлежность набору значений (к примеру, пол «Ж» или «М»).

Технология оперативной обработки транзакции (OLTP-технология)

OLTP, Online Transaction Processing - обработка транзакций в реальном времени. Способ организации БД, при котором система работает с небольшими по размерам транзакциями, но идущими большим потоком, и при этом клиенту требуется от системы минимальное время отклика.

Термин OLTP применяют также к системам (приложениям). OLTP-системы предназначены для ввода, структурированного хранения и обработки информации (операций, документов) в режиме реального времени.

OLTP-приложениями охватывается широкий спектр задач во многих отраслях - автоматизированные банковские системы, ERP-системы (системы планирования ресурсов предприятия), банковские и биржевые операции, в промышленности - регистрация прохождения детали на конвейере, фиксация в статистике посещений очередного посетителя веб-сайта, автоматизация бухгалтерского, складского учёта и учёта документов и т. п.

Приложения OLTP, как правило, автоматизируют структурированные, повторяющиеся задачи обработки данных, такие как ввод заказов и банковские транзакции. OLTP-системы проектируются, настраиваются и оптимизируются для выполнения максимального количества транзакций за короткие промежутки времени. Как правило, большой гибкости здесь не требуется, и чаще всего используется фиксированный набор надёжных и безопасных методов ввода, модификации, удаления данных и выпуска оперативной отчётности. Показателем эффективности является количество транзакций, выполняемых за секунду. Обычно аналитические возможности OLTP-систем сильно ограничены либо вообще отсутствуют.

Требованиями OLTP-обработки информации являются:

- строго нормализованные модели данных;
- при возникновении ошибки транзакция должна целиком откатиться и вернуть систему к состоянию, которое было до начала транзакции;
- обработка данных в реальном времени (с малой задержкой).

Преимуществами OLTP-обработки являются высокая надёжность и достоверность данных, как следствие транзакционного подхода. Транзакция либо совершается полностью и успешно, либо не совершается и система возвращается к предыдущему состоянию. При любом исходе выполнения транзакции целостность данных не нарушается.

Недостатками OLTP-систем является оптимизация только к небольшим дискретным транзакциям. Запросы на некую комплексную информацию, к примеру поквартальная динамика

объемов продаж по определённой модели товара в определённом филиале, характерные для аналитических приложений (OLAP), порождают сложные соединения таблиц и просмотр таблиц целиком. На один такой запрос затрачивается масса времени и компьютерных ресурсов, что тормозит обработку текущих транзакций до неприемлемого уровня.

Информационные хранилища

Информационное хранилище (data warehouse) – это автоматизированная система, которая собирает данные из существующих внутренних баз предприятия и внешних источников, формирует, хранит и эксплуатирует информацию как единую.

Информационное хранилище обеспечивает доступ как к внутренним данным предприятия, так и к внешним источникам данных, доступных по Интернету.

Информационные хранилища поддерживают большое число моделей данных, включая многомерные и ER-модели, что обеспечивает исторические запросы (запросы за прошлые годы и десятилетия), запросы как к оперативным данным предприятия, так и к данным внешних источников, запросы аналитических (агрегированных) данных для анализа тенденций и принятия стратегических решений.

Данные информационных хранилищ хранятся не только на сервере, но и на вторичных устройствах хранения.

При слиянии данных из разных источников и размещении их в информационном хранилище обеспечивается:

- Предметная ориентация. Данные организованы в соответствии со способом их представления в предметных приложениях. В отличие от локальных баз информационное хранилище содержит агрегированные данные приложений и не содержат ненужной с точки зрения анализа информации, что значительно сокращает объемы хранимой информации.

- Целостность и внутренняя взаимосвязь. Хотя данные погружаются из разных внутренних и внешних источников, они объединены едиными законами наименования, способами измерения размерностей и т.д. В разных источниках одинаковые по наименованию данные могут иметь разные формы представления (например, даты) или названия (например, «вероятность доведения информации» в одном источнике и «вероятность получения информации» – в другом). Подобные несоответствия удаляются автоматически.

- Отсутствие временной привязки. Оперативные базы предприятия содержат данные за небольшой интервал времени (неделя, месяц), что достигается за счет периодического архивирования данных. Информационное хранилище содержит исторические данные, накопленные за большой интервал времени (года, десятилетия).

- Упорядоченность во времени; данные согласуются во времени (например, приводятся к единому курсу рубля на текущий момент) для использования в сравнениях, трендах и прогнозах.

- Неизменяемость. Данные не обновляются и не изменяются, а только перезагружаются и считываются из источников на сервер, поддерживая концепцию «одного правдивого источника». Данные доступны только для чтения, так как их модификация может привести к нарушению целостности данных хранилища. Таким образом, данные, погруженные в хранилище, организуясь в интегрированную целостную структуру, обладающую естественными внутренними связями, приобретают новые свойства, придающие им статус информации. Они являются основой для построения аналитических систем и систем поддержки принятия решений. Именно поэтому технологии информационных хранилищ ориентированы на руководителей, ответственных за принятие решений.

OLAP-технология

OLAP (On-Line Analytical Processing) –

1. это класс приложений и технологий, предназначенных для оперативной аналитической обработки многомерных данных (сбор, хранение, анализ) для анализа деятельности корпорации и прогнозирования будущего состояния с целью поддержки принятия управленческих решений.

2. набор технологий для оперативной обработки информации, включающих динамическое построение отчётов в различных разрезах, анализ данных, мониторинг и прогнозирование ключевых показателей бизнеса.

Технология OLAP применяется, чтобы упростить работу с многоцелевыми накопленными данными о деятельности корпорации в прошлом и не погрязнуть в их большом объеме, а также превратить набор количественных показателей в качественные, позволяет аналитикам, менеджерам и управляющим сформировать свое собственное видение данных, используя быстрый, единообразный, оперативный доступ к разнообразным формам представления информации. Такие формы, полученные на основании первичных данных, позволяют пользователю сформировать полноценное представление о деятельности предприятия.

OLAP-технология является альтернативой традиционным методам анализа данных, основанным на различных системах реализации SQL-запросов к реляционной БД.

В основе OLAP-технологий лежит представление информации в виде OLAP-кубов.

OLAP-кубы содержат бизнес-показатели, используемые для анализа и принятия управленческих решений, например: прибыль, рентабельность продукции, совокупные средства (активы), собственные средства, заемные средства и т.д.

Как и любая технология OLAP также имеет свои недостатки: высокие требования к аппаратному обеспечению, подготовке и знаниям административного персонала и конечных пользователей, высокие затраты на реализацию проекта внедрения (как денежные, так и временные, интеллектуальные).

Заключение: Эффект от правильной организации, стратегического и оперативного планирования развития бизнеса трудно заранее оценить в цифрах, но очевидно, что он в десятки и даже сотни раз может превзойти затраты на реализацию таких систем. Однако не следует и заблуждаться. Эффект обеспечивает не сама система, а люди с ней работающие.

Принципы построения и архитектура компьютерных сетей Протоколы, иерархия протоколов и режимы их работы

Классификация современных компьютерных сетей

Все многообразие компьютерных сетей можно классифицировать по следующим четырём признакам:

1. по типу среды передачи, то есть физической среды, которая используется для соединения компьютеров;
2. по скорости передачи информации;
3. по ведомственной принадлежности;
4. по территориальной распространённости.

1) Среда передачи называется еще "линией связи". Информация передается по линиям связи в виде различных сигналов, которые, испытывая сопротивление среды, затухают с расстоянием. Поэтому одной из важнейших характеристик линии связи является максимальная дальность, на которую может быть передана по ней информация без искажения.

В качестве линий связи могут использоваться:

- ИК-лучи (обеспечивают передачу информации между компьютерами, находящимися в пределах одной комнаты);
- электрические провода (кабель "витая пара" обеспечивает связь между компьютерами на расстояние до 100м, коаксиальные кабели – до 500м);
- оптоволоконные кабели (обеспечивают связь на расстояние нескольких десятков километров);
- телефонные линии, радиосвязь, спутниковая связь (позволяют соединять компьютеры, находящиеся в любой точке планеты).

2) По скорости передачи информации компьютерные сети делятся на **низкоскоростные** (скорость передачи информации до 10 Мбит/с), **среднескоростные** (скорость передачи информации до 100 Мбит/с), **высокоскоростные** (скорость передачи информации свыше 100 Мбит/с).

3) По принадлежности различают **ведомственные** и **государственные** сети. Ведомственные сети принадлежат одной организации и располагаются на ее территории. Государственные сети – это сети, используемые в государственных структурах.

4) По территориальной распространённости сети могут быть локальными, глобальными и региональными. **Локальными** называются сети, расположенные в одном или нескольких зданиях. **Региональными** называются сети, расположенные на территории города или области. **Глобальными** называются сети, расположенные на территории государства или группы государств, например, всемирная сеть Интернет.

В классификации сетей существует два основных термина: **локальная сеть (LAN)** и **территориально-распределенная сеть (WAN)**.

Локальная сеть (Local Area Network) связывает компьютеры и принтеры, обычно находящиеся в одном здании (или комплексе зданий). Каждый компьютер, подключенный к локальной сети, называется **рабочей станцией** или **сетевым узлом**. Как правило, в локальных сетях практикуется использование высокоскоростных каналов.

Локальные сети позволяют отдельным пользователям легко и быстро взаимодействовать друг с другом. Вот лишь некоторые задачи, которые позволяет выполнять локальная сеть:

- совместная работа с документами;
- передача файлов между компьютерами без использования каких-либо носителей;
- упрощение документооборота: вы получаете возможность просматривать, корректировать и комментировать документы, не покидая своего рабочего места, не организовывая собраний и совещаний;
- сохранение и архивирование своей работы на сервере, чтобы не использовать ценное пространство на жестком диске компьютера;
- простой доступ к приложениям на сервере;
- облегчение совместного использования дорогостоящих ресурсов, таких как высокопроизводительные принтеры, пишущие дисковые накопители, профессиональные сканеры, жесткие диски большой емкости и программные приложения (например, текстовые

процессоры или программное обеспечение баз данных).

Локальные вычислительные сети подразделяются на два кардинально различающихся класса: одноранговые (одноуровневые или Peer to Peer) сети и иерархические (многоуровневые).

Одноранговая сеть представляет собой сеть равноправных компьютеров, каждый из которых имеет уникальное имя (имя компьютера) и обычно пароль для входа в сеть во время загрузки операционной системы. Имя и пароль входа назначаются владельцем компьютера. Одноранговые сети могут быть организованы с помощью таких операционных систем, как LANtastic, Windows'3.11, Novell NetWare Lite. Одноранговые сети могут быть организованы также на базе всех современных 32-разрядных операционных систем – Windows'95 OSR2, Windows NT Workstation версии, OS/2) и некоторых других.

В **иерархических локальных сетях** имеется один или несколько специальных компьютеров – серверов, на которых хранится информация, совместно используемая различными пользователями.

Сервер в иерархических сетях – это постоянное хранилище разделяемых ресурсов. Сам сервер может быть клиентом только сервера более высокого уровня иерархии. Поэтому иерархические сети иногда называются сетями с выделенным сервером. Серверы обычно представляют собой высокопроизводительные компьютеры, возможно, с несколькими параллельно работающими процессорами, с винчестерами большой емкости, с высокоскоростной сетевой картой (100 Мбит/с и более). Компьютеры, с которых осуществляется доступ к информации на сервере, называются **клиентами**.

2) Территориально-распределенная сеть (Wide Area Network) соединяет несколько локальных сетей, географически удаленных друг от друга. Территориально-распределенные сети обеспечивают те же преимущества, что и локальные, но при этом позволяют охватить большую территорию. Обычно для этого используется коммутируемая телефонная сеть общего пользования (PSTN, Public Switched Telephone Network) с соединением через модем или линии высокоскоростной цифровой сети с предоставлением комплексных услуг (ISDN, Integrated Services Digital Network). Линии ISDN часто применяются для передачи больших файлов, например содержащих графические изображения или видео.

Сетевые технологии

Ethernet - самая популярная технология построения локальных сетей. Основанная на стандарте IEEE 802.3, Ethernet передает данные со скоростью 10 Мбит/с. В сети Ethernet устройства проверяют наличие сигнала в сетевом канале ("прослушивают" его). Если канал не использует никакое другое устройство, то устройство Ethernet передает данные. Каждая рабочая станция в этом сегменте локальной сети анализирует данные и определяет, предназначены ли они ей.

Технология Клиент-сервер. Характер взаимодействия компьютеров в локальной сети принято связывать с их функциональным назначением. Как и в случае прямого соединения, в рамках локальных сетей используется понятие клиент и сервер. **Технология клиент-сервер** — это особый способ взаимодействия компьютеров в локальной сети, при котором один из компьютеров (**сервер**) предоставляет свои ресурсы другому компьютеру (**клиенту**). В соответствии с этим различают одноранговые сети и серверные сети.

При **одноранговой архитектуре** в сети отсутствуют выделенные серверы, каждая рабочая станция может выполнять функции клиента и сервера. В этом случае рабочая станция выделяет часть своих ресурсов в общее пользование всем рабочим станциям сети. Как правило, одноранговые сети создаются на базе одинаковых по мощности компьютеров.

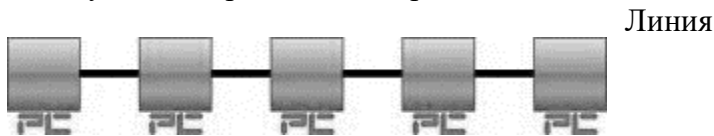
Наличие распределенных данных и возможность изменения своих серверных ресурсов каждой рабочей станцией усложняет защиту информации от несанкционированного доступа, что является одним из недостатков одноранговых сетей. Другим недостатком одноранговых сетей является их более низкая производительность. Это объясняется тем, что сетевые ресурсы сосредоточены на рабочих станциях, которым приходится одновременно выполнять функции клиентов и серверов.

В **серверных сетях** осуществляется четкое разделение функций между компьютерами: одни из них постоянно являются клиентами, а другие — серверами. Учитывая многообразие услуг, предоставляемых компьютерными сетями, существует несколько типов серверов, а именно: сетевой сервер, файловый сервер, сервер печати, почтовый сервер и др.

Сетевой сервер представляет собой специализированный компьютер, ориентированный на выполнение основного объема вычислительных работ и функций по управлению компьютерной сетью.

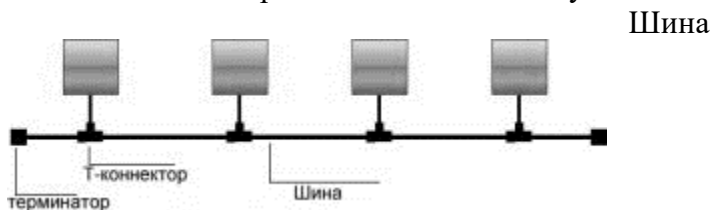
Принципы построения и архитектура компьютерных сетей

Организация обмена данными в сфере компьютерных и информационных технологий осуществляется согласно выбранной топологии. Конфигурация которой определяется соединением нескольких компьютеров и может отличаться от конфигурации логической связи. Выбор типа физической связи непосредственно влияет на характеристики сети, поэтому к данному процессу подходят с учетом определенных требований.



Концепция формирования данного типа сети основана на принципе размещения всех абонентов на одной линии, поэтому при ее повреждении вся цепочка становится неработоспособной, точно так же как и при выключении одного компьютера, когда теряется связующая нить между всеми пользователями.

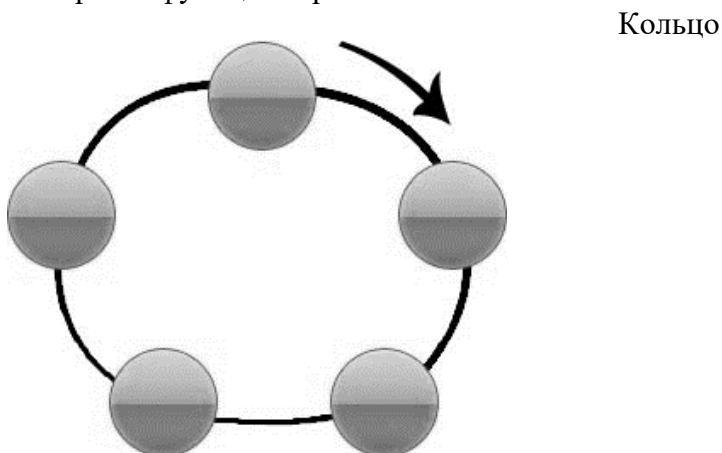
Учитывая несовершенство и моральное устаревание данного типа построения ЛВС, на сегодняшний день его практически не используют.



В данном случае задействован единый кабель, к которому через специальные соединительные элементы подключены ПК. Концы шины снабжены резисторами, препятствующими отражению сигнала и гарантирующими его чистоту.

Преимущество такой топологии состоит в простоте монтажа и настройки, при этом затрачивается меньшее количество кабеля, нежели в других типах сетей. При поломке одного компьютера сеть сохраняет рабочее состояние, однако при неполадках в самой сети функционирование в ее рамках прекращается абсолютно для всех абонентов.

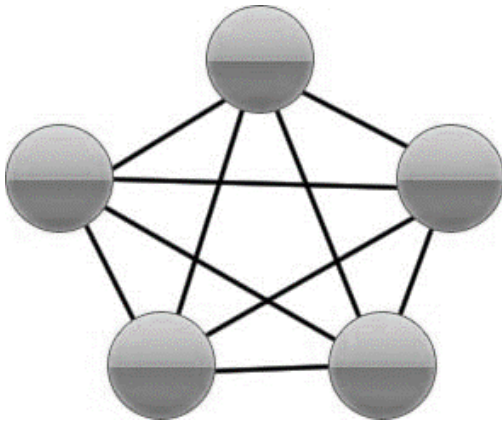
Стоит помнить, что чем больше рабочих станций локализуется на шине, тем существеннее падает скорость функционирования сети.



Общие принципы построения компьютерных сетей по типу кольца аналогичны тем, что применяются при создании топологии линии, однако существуют некоторые отличия, и наиболее существенное из них состоит в последовательном подключении компьютеров друг к другу. Сигнал в такой сети перемещается исключительно в одностороннем порядке, а для обеспечения движения двух сигналов в разных направлениях формируют двойное кольцо.

Данная сеть проста в сборке и не требует большого количества оборудования, при этом она демонстрирует устойчивую работу, однако при неполадках в функционировании одного из ПК вся система оказывается нерабочей.

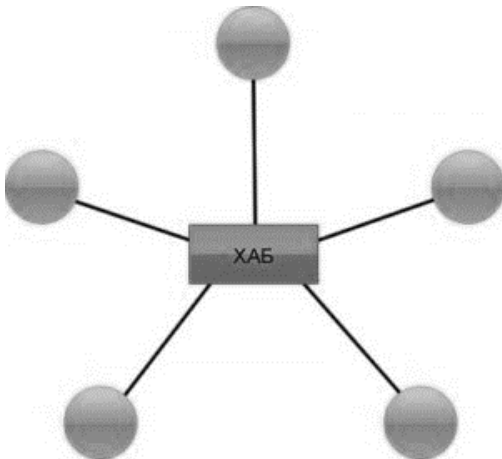
Многосвязная



Преимущество многосвязной конфигурации – высокая скорость обмена файлами, к тому же при поломке одного компьютера другие участники процесса могут и далее осуществлять бесперебойную работу в сети.

Ввиду дороговизны такая сеть применяется очень редко и только там, где необходима высокая скорость и повышенная надежность работы (стратегические объекты).

Звезда



В данном случае не нужно использовать много кабеля и дополнительные спецсредства, однако все абоненты могут быть удалены от концентратора (хаба) не далее чем на 100 метров. Разумеется, при выходе из строя хаба все компьютеры лишаются соединения, однако при поломке одного компьютера или отдельного канала связи сеть продолжает нормально функционировать.

Архитектура сети – это набор параметров, правил, протоколов, алгоритмов, карт, которые позволяют изучать сеть.

Архитектура сети определяет основные элементы сети, характеризует ее общую логическую организацию, техническое обеспечение, программное обеспечение, описывает методы кодирования. Архитектура также определяет принципы функционирования и интерфейс пользователя.

Рассмотрим три вида архитектур:

- архитектура терминал-главный компьютер;
- одноранговая архитектура;
- архитектура клиент-сервер.

Архитектура терминал-главный компьютер – это концепция информационной сети, в которой вся обработка данных осуществляется одним или группой главных компьютеров. Классический пример архитектуры сети с главными компьютерами – системная сетевая архитектура

Одноранговая архитектура – это концепция информационной сети, в которой ее ресурсы рассредоточены по всем системам. Данная архитектура характеризуется тем, что в ней все системы равноправны. Проблемой одноранговой архитектуры является ситуация, когда компьютеры

отключаются от сети. В этих случаях из сети исчезают виды сервиса, которые они предоставляли. Существенным недостатком одноранговых сетей является отсутствие централизованного администрирования. Хорошо подходят для сетей с количеством пользователей, не превышающим десяти

Архитектура клиент-сервер (client-server architecture) – это концепция информационной сети, в которой основная часть ее ресурсов сосредоточена в серверах, обслуживающих своих клиентов. Рассматриваемая архитектура определяет два типа компонентов: серверы и клиенты. Сервер – это объект, предоставляющий сервис другим объектам сети по их запросам. Сервис – это процесс обслуживания клиентов. Клиенты – это рабочие станции, которые используют ресурсы сервера и предоставляют удобные интерфейсы пользователя. Сети на базе серверов имеют лучшие характеристики и повышенную надежность. Сервер владеет главными ресурсами сети, к которым обращаются остальные рабочие станции.

Выбор архитектуры сети зависит от назначения сети, количества рабочих станций и от выполняемых на ней действий.

Для стандартизации сетей Международная организация стандартов (OSI) предложила семиуровневую сетевую архитектуру. К сожалению, конкретные реализации сетей не используют все уровни международного стандарта. Однако этот стандарт дает общее представление о взаимодействии отдельных подсистем сети.

Семиуровневая сетевая архитектура

7	Прикладной уровень (Application Layer).
6	Представительный уровень (Presentation Layer).
5	Сеансовый уровень (Session Layer).
4	Транспортный уровень (Transport Layer).
3	Сетевой уровень (Network Layer).
2	Канальный уровень (data Link).
1	Физический уровень (Physical Layer).

Физический уровень (Physical Layer)

Физический уровень (Physical layer) имеет дело с передачей битов по физическим каналам связи, таким, например, как коаксиальный кабель, витая пара, оптоволоконный кабель или цифровой территориальный канал. К этому уровню имеют отношение характеристики физических сред передачи данных, такие как полоса пропускания, помехозащищенность, волновое сопротивление и другие. На этом же уровне определяются характеристики электрических сигналов, передающих дискретную информацию, например, крутизна фронтов импульсов, уровни напряжения или тока передаваемого сигнала, тип кодирования, скорость передачи сигналов. Кроме этого, здесь стандартизируются типы разъемов и назначение каждого контакта.

Функции физического уровня реализуются во всех устройствах, подключенных к сети. Со стороны компьютера функции физического уровня выполняются сетевым адаптером или последовательным портом.

Канальный уровень (data Link)

На физическом уровне просто пересылаются биты. При этом не учитывается, что в некоторых сетях, в которых линии связи используются (разделяются) попеременно несколькими парами взаимодействующих компьютеров, физическая среда передачи может быть занята. Поэтому одной из задач канального уровня (Data Link layer) является проверка доступности среды передачи. Другой задачей канального уровня является реализация механизмов обнаружения и коррекции ошибок. Для этого на канальном уровне биты группируются в наборы, называемые *кадрами (frames)*. Канальный уровень обеспечивает корректность передачи каждого кадра, помещая специальную последовательность бит в начало и конец каждого кадра, для его выделения, а также вычисляет контрольную сумму, обрабатывая все байты кадра определенным способом и добавляя контрольную сумму к кадру. Когда кадр приходит по сети, получатель снова вычисляет контрольную сумму полученных данных и сравнивает результат с контрольной суммой из кадра. Если они совпадают, кадр считается правильным и принимается. Если же контрольные суммы не

совпадают, то фиксируется ошибка. Канальный уровень может не только обнаруживать ошибки, но и исправлять их за счет повторной передачи поврежденных кадров. Необходимо отметить, что функция исправления ошибок не является обязательной для канального уровня, поэтому в некоторых протоколах этого уровня она отсутствует, например, в Ethernet и frame relay.

В локальных сетях протоколы канального уровня используются компьютерами, мостами, коммутаторами и маршрутизаторами. В компьютерах функции канального уровня реализуются совместными усилиями сетевых адаптеров и их драйверов.

Сетевой уровень (Network Layer)

Сетевой уровень (Network layer) служит для образования единой транспортной системы, объединяющей несколько сетей, причем эти сети могут использовать совершенно различные принципы передачи сообщений между конечными узлами и обладать произвольной структурой связей.

Доставкой данных между сетями занимается сетевой уровень, который и поддерживает возможность правильного выбора маршрута передачи сообщения даже в том случае, когда структура связей между составляющими сетями имеет характер, отличный от принятого в протоколах канального уровня. Сети соединяются между собой специальными устройствами, называемыми маршрутизаторами. *Маршрутизатор* - это устройство, которое собирает информацию о топологии межсетевых соединений и на ее основании пересылает пакеты сетевого уровня в сеть назначения.

Чтобы передать сообщение от отправителя, находящегося в одной сети, получателю, находящемуся в другой сети, нужно совершить некоторое количество *транзитных* передач между сетями, или хопов (от hop - прыжок), каждый раз выбирая подходящий маршрут. Таким образом, маршрут представляет собой последовательность маршрутизаторов, через которые проходит пакет.

Проблема выбора наилучшего пути называется *маршрутизацией*, и ее решение является одной из главных задач сетевого уровня. Эта проблема осложняется тем, что самый короткий путь не всегда самый лучший. Часто критерием при выборе маршрута является время передачи данных по этому маршруту; оно зависит от пропускной способности каналов связи и интенсивности трафика, которая может изменяться с течением времени. Некоторые алгоритмы маршрутизации пытаются приспособиться к изменению нагрузки, в то время как другие принимают решения на основе средних показателей за длительное время. Выбор маршрута может осуществляться и по другим критериям, например надежности передачи.

В общем случае функции сетевого уровня шире, чем функции передачи сообщений по связям с нестандартной структурой, которые мы сейчас рассмотрели на примере объединения нескольких локальных сетей. Сетевой уровень решает также задачи согласования разных технологий, упрощения адресации в крупных сетях и создания надежных и гибких барьеров на пути нежелательного трафика между сетями.

Транспортный уровень (Transport Layer)

Обеспечивает приложениям или верхним уровням стека - прикладному и сеансовому - передачу данных с той степенью надежности, которая им требуется. Модель OSI определяет пять классов сервиса, предоставляемых транспортным уровнем. Эти виды сервиса отличаются качеством предоставляемых услуг: срочностью, возможностью восстановления прерванной связи, наличием средств мультимплексирования нескольких соединений между различными прикладными протоколами через общий транспортный протокол, а главное - способностью к обнаружению и исправлению ошибок передачи, таких как искажение, потеря и дублирование пакетов.

Как правило, все протоколы, начиная с транспортного уровня и выше, реализуются программными средствами конечных узлов сети - компонентами их сетевых операционных систем. В качестве примера транспортных протоколов можно привести протоколы TCP и UDP стека TCP/IP и протокол SPX стека Novell. Протоколы нижних четырех уровней обобщенно называют сетевым транспортом или транспортной подсистемой, так как они полностью решают задачу транспортировки сообщений с заданным уровнем качества в составных сетях с произвольной топологией и различными технологиями. Остальные три верхних уровня решают задачи предоставления прикладных сервисов на основании имеющейся транспортной подсистемы.

Сеансовый уровень (Session Layer)

Обеспечивает управление диалогом: фиксирует, какая из сторон является активной в настоящий момент, предоставляет средства синхронизации. Последние позволяют вставлять контрольные точки в длинные передачи, чтобы в случае отказа можно было вернуться назад к последней контрольной точке, а не начинать все с начала. В частности, на этом уровне выполняется управление доступом на основе прав доступа.

Представительный уровень (Presentation Layer)

Имеет дело с формой представления передаваемой по сети информации, не меняя при этом ее содержания. За счет уровня представления информация, передаваемая прикладным уровнем одной системы, всегда понятна прикладному уровню другой системы. С помощью средств данного уровня протоколы прикладных уровней могут преодолеть синтаксические различия в представлении данных или же различия в кодах символов, например кодов ASCII и EBCDIC. На этом уровне может выполняться шифрование и дешифрование данных, благодаря которому секретность обмена данными обеспечивается сразу для всех прикладных служб.

Прикладной уровень (Application Layer)

Это в действительности просто набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к разделяемым ресурсам, таким как файлы, принтеры или гипертекстовые Web-страницы, а также организуют свою совместную работу, например, с помощью протокола электронной почты.

Протоколы, иерархия протоколов и режимы их работы

Под сетевым протоколом обычно понимают совокупность правил взаимодействия двух элементов сети при обмене информацией между ними.

Большинство протоколов имеют иерархический принцип организации (рис. 1.3): нижележащий уровень предоставляет через интерфейс некоторый набор услуг (сервисов) вышележащему уровню, не раскрывая детали. В реализации предоставляемой услуги.

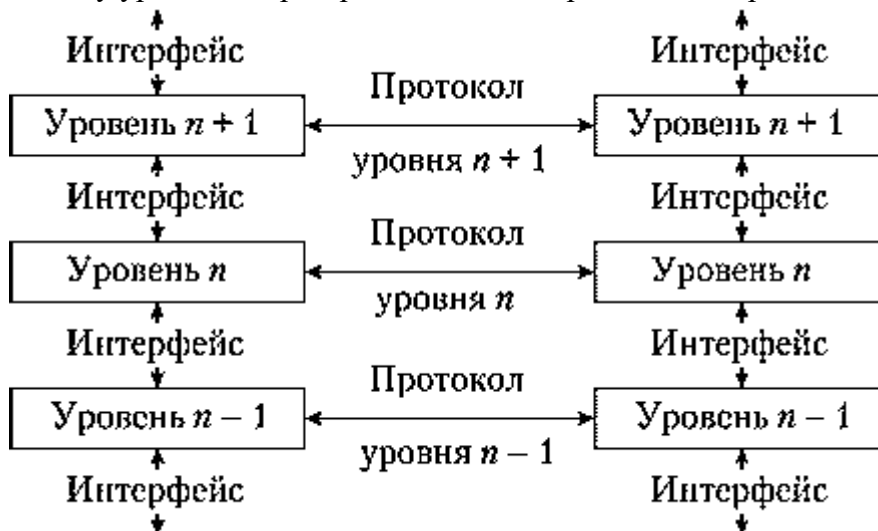


Рис. 1.3. Уровни протоколов [1; 2]

Правила и соглашения, используемые при взаимодействии уровня n одного узла и уровня n другого узла, называются *протоколом уровня n* .

На каждом уровне имеется активный элемент (сущность (Entity)). *Одноранговыми сущностями* называют сущности одного уровня на разных узлах сети. Сущности уровня n предоставляют услуги уровню $n + 1$, т.е. являются поставщиками услуг. При этом сущности уровня $n + 1$ являются потребителями услуг уровня n . Оказывая услуги уровню $n + 1$, уровень n может использовать услуги (быть потребителем) нижележащего уровня $n - 1$.

Сервис (услуга) определяет операции, которые будет выполнять уровень, но не реализацию этих операций. Сервис, по сути, описывает интерфейс взаимодействия двух смежных уровней — поставщика услуг (нижележащего уровня) и потребителя (вышележащего уровня).

Доступ к услугам некоторого уровня обеспечивается через *точки доступа к услуге (Service Access Point, SAP)*, т.е. посредством набора правил используемого для взаимодействия между уровнями интерфейса. Через SAP сущность вышележащего уровня передает сущности нижележащего уровня *элемент данных интерфейса (Interface Data Unit, IDU)*, который состоит из *элемента данных услуги (Service Data Unit, SDU)* и некоторой *управляющей информации (Interface Control Information, ICI)*. При этом SDU может быть разбит на несколько фрагментов. Тогда его пересылка осуществляется в виде отдельных *элементов данных протокола (Protocol Data Unit, PDU)* или *пакетов*.

Таким образом, *протокол* определяет набор правил, которые описывают формат и назначение пакетов, передаваемых между одноранговыми сущностями внутри уровня. По сути, протокол определяет услуги уровня, на котором он работает. Протокол может претерпеть изменения, но предоставляемые услуги не должны меняться.

Протоколы разделяют на протоколы с *установлением соединения (Connection Oriented)* и без установления соединения (*Connectionless*). В первом случае до начала обмена данными отправитель и получатель должны установить соединение, определив предварительно некоторые параметры протокола. После завершения передачи данных (завершения сеанса передачи) соединение должно быть разорвано с помощью обмена специальными управляющими сообщениями.

Во втором случае передача данных осуществляется без организации специальной процедуры по установлению соединения с получателем отправляемых данных.

Список протоколов, который использует элемент сети, называется *стеком протоколов*. Совокупность уровней и протоколов образуют архитектуру сети.

Преимущества: Уровни обычно уменьшают сложность связи между сетями. Это увеличивает срок службы сети.

Режим передачи означает передачу данных между двумя устройствами. Он также известен как режим связи. Шины и сети предназначены для обеспечения связи между отдельными устройствами, которые связаны между собой. Существует три типа режима передачи: симплексный, полудуплексный и полнодуплексный.

В симплексном режиме связь однонаправленная, как на улице с односторонним движением. Только одно из двух устройств на линии может передавать, другое может только принимать. Симплексный режим может использовать всю емкость канала для передачи данных в одном направлении.

Пример: клавиатура и традиционные мониторы. Клавиатура может вводить только ввод, монитор может выдавать только вывод.

В полудуплексном режиме каждая станция может как передавать, так и принимать, но не одновременно. Когда одно устройство отправляет, другое может только получать, и наоборот. Полудуплексный режим используется в тех случаях, когда нет необходимости в связи в обоих направлениях одновременно. Для каждого направления может быть использована вся пропускная способность канала.

Пример: Рация, в которой сообщение отправляется по одному, а сообщения отправляются в обоих направлениях.

В полнодуплексном режиме обе станции могут передавать и принимать одновременно. В режиме *full_duplex* сигналы, идущие в одном направлении, делят пропускную способность канала с сигналами, идущими в другом направлении, это совместное использование может происходить двумя способами:

Либо ссылка должна содержать два физически отдельных пути передачи, один для отправки, а другой для приема.

Или емкость делится между сигналами, движущимися в обоих направлениях.

Полнодуплексный режим используется, когда связь в обоих направлениях требуется постоянно. Однако пропускная способность канала должна быть разделена между двумя направлениями.

Пример: Телефонная сеть, в которой имеется связь между двумя лицами по телефонной линии, через которую оба могут одновременно говорить и слушать.

Соединение, передача данных, разъединение

В сеансе связи различают три фазы: установление соединения, передачу данных и разъединение соединения (рис. 5.40).

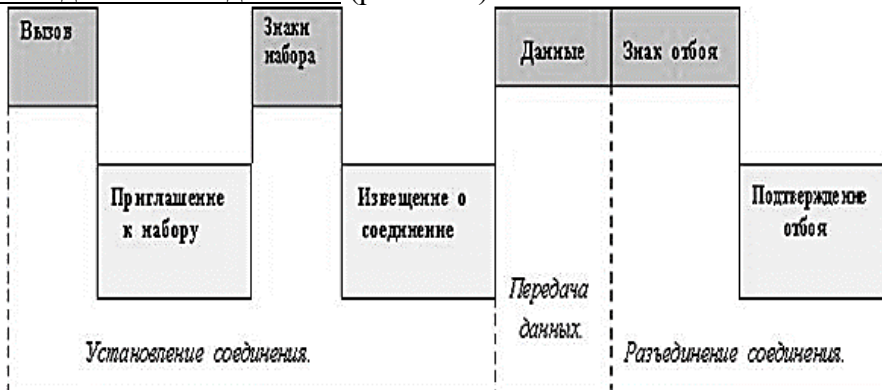


Рис. 5.40. Фазы сеанса связи

Процессом установления соединения управляет источник, который посылает сигнал вызова, получает ответный сигнал (приглашение к набору номера) и вслед за этим передаёт адресную информацию (знаки набора номера). Коммутационный узел обрабатывает эту информацию, занимает один из каналов в пучке, ведущем к следующему коммутационному узлу, и передает последнему знаки набора, необходимые для дальнейшего установления соединения. Таким образом, постепенно, по участкам, вплоть до вызываемого абонента образуется соединительный тракт. После завершения этого процесса, от сети на вызывающую и вызываемую оконечные установки поступают сигналы, извещающие о том, что соединение готово к передаче данных.

В течение фазы передачи данных управление осуществляется оконечной установкой. В оконечной установке принимается решение о мерах, которые необходимо принять для обнаружения и исправления ошибок передачи.

Разъединение может быть начато любой из двух связанных между собой оконечных установок с помощью сигнала отбоя. По этому сигналу все коммутационные узлы, участвующие в образовании соединительного тракта, отключают соединения. Среди сетей передачи данных с коммутацией каналов различают два типа: синхронные и асинхронные сети.

В асинхронных сетях общая синхронизация по элементам отсутствует, и для сети не задаются единые «такты». Отдельные аппаратура передачи данных и коммутационные устройства имеют самостоятельные, независимые друг от друга тактовые генераторы.

В синхронной сети с коммутацией каналов ход во времени всех процессов передачи и коммутации определяется единым тактовым синхросигналом. Он подводится ко всей аппаратуре и оборудованию сети, задаёт для всей сети жесткий временной растр и обеспечивает синхронизм всех процессов.

Передача информации в компьютерных сетях

1. Последовательный способ передачи информации
2. Параллельный способ передачи информации
3. Пакетный принцип организации данных

Последовательный и параллельный способы передачи информации

Информация в компьютерах представлена в форме последовательностей двоичных чисел. Обмен данными как внутри вычислительного устройства между его узлами, так и между автономными машинами, может производиться двумя способами:

Последовательная передача, имеется только одна линия, состояние на ее передающей стороне отправляется только тогда, когда предыдущее обработано принимающей, т.е. данные передаются побитно,

Параллельная передача, при таком способе организуются сразу несколько линий, состояние

на концах которых меняется одновременно, таким образом, можно передать за один раз столько бит, сколько имеется линий между передатчиком и приемником.

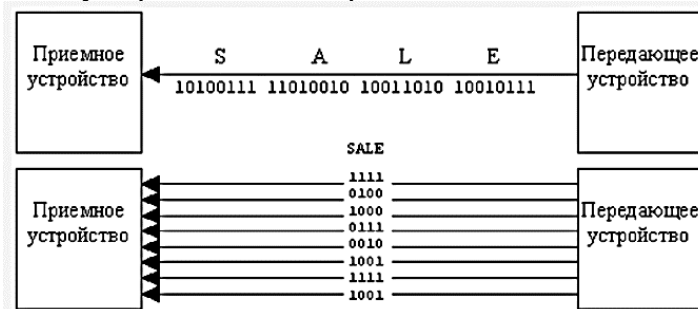


Рис. Последовательная и параллельная передача данных

При параллельной передаче технологически трудно избежать взаимовлияния токов, протекающих по близко расположенным проводникам. Поэтому такой способ используется там, где расстояния невелики: между узлами компьютера (т.н. шина данных), между компьютером и монитором (VGA-порт), между компьютером и принтером (параллельный порт).

Последовательная передача, хотя и уступает параллельной по скорости, обеспечивает более эффективную обработку ошибок и менее затратна в случае отправки данных на большие расстояния: двужильный кабель дешевле и надежнее многожильного.

Для передачи информации в компьютерных сетях в подавляющем большинстве случаев используется последовательная передача данных. Хотя с развитием технологий стало возможным одновременно передавать несколько потоков (разнесение по частотам в wifi, передача по оптоволокну лучей с разным углом наклона), такие способы нельзя назвать параллельной передачей, т.к. данные в каждой такой линии обрабатываются независимо друг от друга.

Пакетный принцип организации данных

При последовательной передаче данные в сетях принято передавать не непрерывным потоком, а пакетами (порциями, сериями). Такой подход обладает следующими преимуществами:

по одной и той же линии можно передавать данные для нескольких получателей, указывая их адреса в заголовочной части пакетов;

получив определенный объем информации, можно убедиться, что содержащиеся в них данные точно соответствуют тому, что было отправлено; для этого в последовательность пакетов добавляются так называемые контрольные суммы - особым образом подсчитанные числа, на которые влияет каждый бит переданной информации; если хотя бы один бит на стороне приемника будет отличаться (например, из-за помех на линии), то контрольные суммы приемника и передатчика не совпадут и станет понятно, что информация принята с искажениями, следует повторить ее отправку/прием.

Пакетный принцип положен в основу протоколов (правил обмена информацией), используемых в современных компьютерных сетях. В большинстве из них используется семейство TCP/IP - набор протоколов для обмена данными в глобальной сети Интернет, представляющей собой объединение локальных сетей.

Каналы связи, модемы

Модем-устройство, применяющееся в системах связи для физического сопряжения информационного сигнала со средой его распространения, где он не может существовать без адаптации. Модулятор в модеме осуществляет модуляцию несущего сигнала при передаче данных, демодулятор осуществляет обратный процесс при приеме данных из канала связи. Модемы широко применяются для связи комп через телефонную сеть. Виды комп модемов: По исполнению: внешние, внутренние, встроенные; По типу сети и соединения: модемы для телефонных линий, кабельные модемы, радиомодемы, беспроводные модемы, спутниковые модемы.

Каналы связи.

Кабельные. Витая пара - кабель связи, который представляет собой витую пару медных проводов, заключенных в экранированную оболочку. Пары проводов скручиваются между собой с целью уменьшения наводок. Скорость передачи данных 10 Мбит/с и 100 Мбит/с. Витая пара используется для связи на расстояние не более нескольких сот метров. Оптоволоконный кабель -

оптическое волокно на кремниевой или пластмассовой основе, заключенное в материал с низким коэффициентом преломления света, который закрыт внешней оболочкой. Скорость передачи данных 3 Гбит/с.

Радиоканалы наземной и спутниковой связи образуются с помощью передатчика и приемника радиоволн и относятся к технологии беспроводной передачи данных. Радиорелейные каналы связи состоят из последовательности станций, являющихся ретрансляторами. Связь осуществляется в пределах прямой видимости, дальности между соседними станциями - до 50 км. В спутниковых системах используются антенны СВЧ-диапазона частот для приема радиосигналов от наземных станций и ретрансляции этих сигналов обратно на наземные станции. Целесообразнее использовать спутниковую связь для организации канала связи между станциями, расположенными на очень больших расстояниях, и возможности обслуживания абонентов в самых труднодоступных точках. Пропускная способность высокая – несколько десятков Мбит/с. Радиоканалы сотовой связи строятся по тем же принципам, что и сотовые телефонные сети. Радиоканалы передачи данных MMDS. Эти системы способна обслуживать территорию в радиусе 50—60 км, при этом прямая видимость передатчика оператора является не обязательной. Средняя гарантированная скорость передачи данных составляет 500 Кбит/с-1 Мбит/с, но можно обеспечить до 56 Мбит/с на один канал. Радиоканалы передачи данных для локальных сетей. Стандартом беспроводной связи для локальных сетей является технология Wi-Fi. Скорость обмена данными до 11 Мбит/с при подключении точка-точка (для подключения двух ПК) и до 54 Мбит/с при инфраструктурном соединении (для подключения несколько ПК к одной точке доступа). Радиоканалы передачи данных Bluetooth-это технология передачи данных на короткие расстояния (не более 10 м) и мб использована для созд домашних сетей. Скорость передачи данных не превышает 1 Мбит/с.

Электронная почта(E-mail) является самой популярной и распространенной службой Internet. Для того чтобы иметь возможность обмениваться письмами по электронной почте, пользователь должен стать клиентом одной из компьютерных сетей. Все письма, поступающие на некоторый почтовый адрес, записываются в выделенную для него область памяти сетевого компа. Сетевой комп, содержащий почтовые ящики абонентов носит название хост компа.

Кодирование и защита от ошибок

Существует три наиболее распространенных орудия борьбы с ошибками в процессе передачи данных:

- коды обнаружения ошибок;
- коды с коррекцией ошибок – схемы прямой коррекции ошибок (*Forward Error Correction - FEC*);
- протоколы с автоматическим запросом повторной передачи (*Automatic Repeat Request - ARQ*).

Код обнаружения ошибок позволяет довольно легко установить наличие ошибки. Как правило, подобные коды используются совместно с определенными протоколами канального или транспортного уровней, имеющими схему *ARQ*. В схеме *ARQ* приемник попросту отклоняет блок данных, в котором была обнаружена ошибка, после чего передатчик передает этот блок повторно. Коды с прямой коррекцией ошибок позволяют не только обнаружить ошибки, но и исправить их, не прибегая к повторной передаче. Схемы *FEC* часто используются в беспроводной передаче, где повторная передача крайне неэффективна, а уровень ошибок довольно высок.

1) Методы обнаружения ошибок

Методы обнаружения ошибок основаны на передаче в составе блока данных избыточной служебной информации, по которой можно судить с некоторой степенью вероятности о достоверности принятых данных.

Избыточную служебную информацию принято называть контрольной суммой, или контрольной последовательностью кадра (*Frame Check Sequence, FCS*). Контрольная сумма вычисляется как функция от основной информации, причем не обязательно путем суммирования. Принимающая сторона повторно вычисляет контрольную сумму кадра по известному алгоритму и в случае ее совпадения с контрольной суммой, вычисленной передающей стороной, делает вывод о том, что данные были переданы через сеть корректно.

2) Методы коррекции ошибок

Наиболее часто в современных системах связи применяется тип кодирования, реализуемый сверхточным кодирующим устройством (*Convolutional coder*), потому что такое кодирование несложно реализовать аппаратно с использованием линий задержки (*delay*) и сумматоров. В отличие от рассмотренного выше кода, который относится к блочным кодам без памяти, сверхточный код относится к кодам с конечной памятью (*Finite memory code*); это означает, что выходная последовательность *кодера* является функцией не только текущего входного сигнала, но также нескольких из числа последних предшествующих битов. Длина кодового ограничения (*Constraint length of a code*) показывает, как много выходных элементов выходит из системы в пересчете на один входной. Коды часто характеризуются их эффективной степенью (или коэффициентом) кодирования (*Code rate*). Вам может встретиться сверхточный код с коэффициентом кодирования $1/2$. Этот коэффициент указывает, что на каждый входной бит приходится два выходных. Хотя коды с более высокой эффективной степенью кодирования позволяют передавать данные с более высокой скоростью, они, соответственно, более чувствительны к шуму.

В беспроводных системах с блочными кодами широко используется метод чередования блоков. Преимущество чередования состоит в том, что приемник распределяет пакет ошибок, исказивший некоторую последовательность битов, по большому числу блоков, благодаря чему становится возможным исправление ошибок. Чередование выполняется с помощью чтения и записи данных в разном порядке. Если во время передачи пакет помех воздействует на некоторую последовательность битов, то все эти биты оказываются разнесенными по различным блокам. Следовательно, от любой контрольной последовательности требуется возможность исправить лишь небольшую часть от общего количества инвертированных битов.

3) Методы автоматического запроса повторной передачи

В простейшем случае защита от ошибок заключается только в их обнаружении. Система должна предупредить передатчик об обнаружении ошибки и необходимости повторной передачи. Такие процедуры защиты от ошибок известны как методы автоматического запроса повторной передачи (*Automatic Repeat Request - ARQ*). В беспроводных локальных сетях применяется процедура "запрос *ARQ* с остановками" (*stop-and-wait ARQ*).

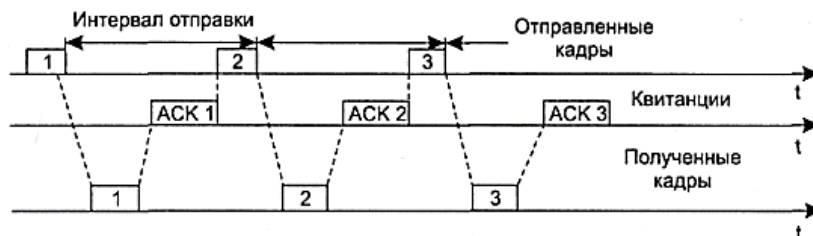


Рис. 1.13. Процедура запрос *ARQ* с остановками

В этом случае источник, пославший кадр, ожидает получения подтверждения (*Acknowledgement - ACK*), или, как еще его называют, квитанции, от приемника и только после этого посылает следующий кадр. Если же подтверждение не приходит в течение тайм-аута, то кадр (или подтверждение) считается утерянным и его передача повторяется. На рис. 1.13 видно, что в этом случае производительность обмена данными ниже потенциально возможной; хотя передатчик и мог бы послать следующий кадр сразу же после отправки предыдущего, он обязан ждать прихода подтверждения.

Структура пакета

Структура и размеры *пакета* в каждой сети жестко определены стандартом на данную *сеть* и связаны, прежде всего, с аппаратными особенностями данной сети, выбранной топологией и типом среды передачи информации. Кроме того, эти параметры зависят от используемого протокола (порядка обмена информацией).

Но существуют некоторые общие принципы формирования структуры *пакета*, которые учитывают характерные особенности обмена информацией по любым локальным сетям.



- Стартовая комбинация битов или преамбула, которая обеспечивает предварительную настройку аппаратуры адаптера или другого сетевого устройства на прием и обработку *пакета*. Это *поле* может полностью отсутствовать или же сводиться к единственному стартовому биту.

- Сетевой адрес (идентификатор) принимающего абонента, то есть индивидуальный или групповой номер, присвоенный каждому принимающему абоненту в сети. Этот адрес позволяет приемнику распознать *пакет*, адресованный ему лично, группе, в которую он входит, или всем абонентам сети одновременно (при широком вещании).

- Сетевой адрес (идентификатор) передающего абонента, то есть индивидуальный номер, присвоенный каждому передающему абоненту. Этот адрес информирует принимающего абонента, откуда пришел данный *пакет*. Включение в *пакет* адреса передатчика необходимо в том случае, когда одному приемнику могут попеременно приходить *пакеты* от разных передатчиков.

- Служебная информация, которая может указывать на тип *пакета*, его номер, размер, формат, маршрут его доставки, на то, что с ним надо делать приемнику и т.д.

- Данные (*поле данных*) – это та информация, ради передачи которой используется *пакет*. В отличие от всех остальных *полей пакета* *поле данных* имеет переменную длину, которая, собственно, и определяет полную длину *пакета*. Существуют специальные управляющие *пакеты*, которые не имеют поля данных. Их можно рассматривать как сетевые команды. *Пакеты*, включающие *поле данных*, называются информационными *пакетами*. Управляющие *пакеты* могут выполнять функцию начала и конца сеанса связи, подтверждения приема информационного *пакета*, запроса информационного *пакета* и т.д.

- Контрольная сумма *пакета* – это числовой код, формируемый передатчиком по определенным правилам и содержащий в свернутом виде информацию обо всем *пакете*. Приемник, повторяя вычисления, сделанные передатчиком, с принятым *пакетом*, сравнивает их результат с контрольной суммой и делает вывод о правильности или ошибочности передачи *пакета*. Если *пакет* ошибочен, то приемник запрашивает его повторную передачу. Обычно используется циклическая контрольная сумма (CRC). Подробнее об этом рассказано в главе 7.

- Стоповая комбинация служит для информирования аппаратуры принимающего абонента об окончании *пакета*, обеспечивает выход аппаратуры приемника из состояния приема. Это *поле* может отсутствовать, если используется *самосинхронизирующий код*, позволяющий определять момент окончания передачи *пакета*.

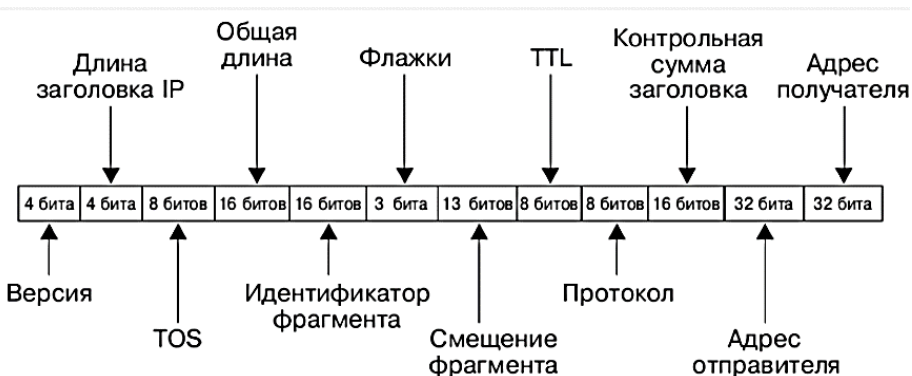


Рис. Структура заголовка IP-пакета

Методы коммутации каналов, сообщений, пакетов

В общем случае решение каждой из частных задач коммутации — *определение* потоков и соответствующих маршрутов, фиксация маршрутов в *конфигурационных параметрах* и таблицах сетевых устройств, *распознавание* потоков и *передача данных* между интерфейсами одного устройства, *мультиплексирование/демультиплексирование* потоков и *разделение среды передачи* — тесно связано с решением всех остальных. Комплекс технических решений обобщенной задачи коммутации в совокупности составляет *базис* любой *сетевой технологии*. От того, какой механизм прокладки маршрутов, продвижения данных и совместного использования каналов связи заложен в той или иной *сетевой технологии*, зависят ее фундаментальные свойства.

Среди множества возможных подходов к решению задачи коммутации абонентов в сетях выделяют два основополагающих:

- *коммутация каналов (circuit switching)*;
- *коммутация пакетов (packet switching)*.

Внешне обе эти схемы соответствуют приведенной на [рис. 6.1](#) структуре сети, однако возможности и свойства их различны.

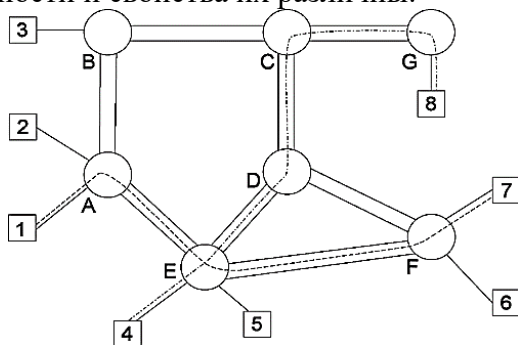


Рис. 6.1. Общая структура сети с коммутацией абонентов

Сети с коммутацией каналов имеют более богатую историю, они произошли от первых *телефонных сетей*. Сети с коммутацией *пакетов* сравнительно молоды, они появились в конце 60-х годов как результат экспериментов с первыми глобальными *компьютерными сетями*. Каждая из этих схем имеет свои достоинства и недостатки, но по долгосрочным прогнозам многих специалистов, будущее принадлежит технологии коммутации *пакетов*, как более гибкой и универсальной.

Коммутация каналов

При *коммутации каналов* *коммутационная сеть* образует между конечными узлами **непрерывный составной физический канал** из последовательно соединенных *коммутаторами* промежуточных канальных участков. Условием того, что несколько физических каналов при последовательном *соединении* образуют единый *физический канал*, является равенство скоростей передачи данных в каждом из составляющих физических каналов. *Равенство* скоростей означает, что *коммутаторы* такой сети не должны буферизовать передаваемые данные.

В *сети с коммутацией каналов* перед передачей данных всегда необходимо выполнить процедуру установления *соединения*, в процессе которой и создается составной канал. И только после этого можно начинать передавать данные.

Например, если *сеть*, изображенная на [рис. 6.1](#), работает по технологии *коммутации каналов*, то узел 1, чтобы передать данные узлу 7, сначала должен передать специальный *запрос* на установление *соединения* коммутатору А, указав *адрес* назначения 7. Коммутатор А должен выбрать *маршрут* образования составного канала, а затем передать *запрос* следующему коммутатору, в данном случае Е. Затем коммутатор Е передает *запрос* коммутатору F, а тот, в свою очередь, передает *запрос* узлу 7. Если узел 7 принимает *запрос* на установление *соединения*, он направляет по уже установленному каналу ответ исходному узлу, после чего составной канал считается скоммутированным, и узлы 1 и 7 могут обмениваться по нему данными.

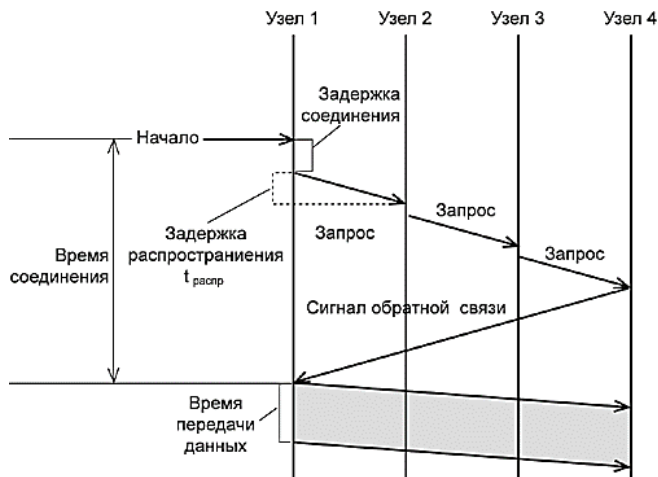


Рис. 6.2. Установление составного канала

Техника *коммутации каналов* имеет свои достоинства и недостатки.

Достоинства *коммутации каналов*

1. Постоянная и известная скорость передачи данных по установленному между конечными узлами каналу. Это дает пользователю сети возможности на основе заранее произведенной оценки необходимой для качественной передачи данных пропускной способности установить в сети канал нужной скорости.

2. **Низкий** и постоянный *уровень задержки* передачи данных через сеть. Это позволяет качественно передавать данные, чувствительные к задержкам (называемые также трафиком реального времени) — голос, видео, различную технологическую информацию.

Недостатки *коммутации каналов*

1. *Отказ сети в обслуживании* запроса на установление *соединения*. Такая ситуация может сложиться из-за того, что на некотором участке сети *соединение* нужно установить вдоль канала, через который уже проходит максимально возможное количество информационных потоков. Отказ может случиться и на конечном участке составного канала — например, если абонент способен поддерживать только одно *соединение*, что характерно для многих *телефонных сетей*. При поступлении второго вызова к уже разговаривающему абоненту сеть передает вызывающему абоненту короткие гудки — сигнал "занято".

2. Нерациональное **использование пропускной способности** физических каналов. Та часть пропускной способности, которая отводится составному каналу после установления *соединения*, предоставляется ему на все время, т.е. до тех пор, пока *соединение* не будет разорвано. Однако абонентам не всегда нужна пропускная способность канала во время *соединения*, например в телефонном разговоре могут быть паузы, еще более неравномерным во времени является взаимодействие компьютеров. Невозможность динамического перераспределения пропускной способности представляет собой принципиальное ограничение *сети с коммутацией каналов*, так как единицей коммутации здесь является *информационный поток* в целом.

3. Обязательная задержка перед передачей данных из-за фазы установления *соединения*.

Достоинства и недостатки любой *сетевой технологии* относительны. В определенных ситуациях на первый план выходят достоинства, а недостатки становятся несущественными. Так, техника *коммутации каналов* хорошо работает в тех случаях, когда нужно передавать только трафик телефонных разговоров. Здесь с невозможностью "вырезать" паузы из разговора и более рационально использовать магистральные физические каналы между *коммутаторами* можно мириться. А вот при передаче очень неравномерного компьютерного трафика эта нерациональность уже выходит на первый план.

Коммутация пакетов

Эта техника коммутации была специально разработана для эффективной передачи компьютерного **трафика**. Первые шаги на пути создания компьютерных сетей на основе техники *коммутации каналов* показали, что этот вид коммутации не позволяет достичь высокой *общей пропускной способности* сети. Типичные *сетевые приложения* генерируют трафик очень неравномерно, с высоким уровнем пульсации **скорости передачи** данных. Например, при обращении к удаленному *файловому серверу* пользователь сначала просматривает содержимое

каталога этого сервера, что порождает передачу небольшого объема данных. Затем он открывает требуемый *файл* в текстовом редакторе, и эта операция может создать достаточно интенсивный *обмен данными*, особенно если *файл* содержит объемные графические включения. После отображения нескольких страниц файла *пользователь* некоторое время работает с ними локально, что вообще не требует передачи данных *по сети*, а затем возвращает модифицированные копии страниц на *сервер* — и это снова порождает интенсивную передачу данных *по сети*.

Коэффициент *пульсации трафика* отдельного пользователя сети, равный отношению средней интенсивности обмена данными к максимально возможной, может достигать 1:50 или даже 1:100. Если для описанной сессии организовать коммутацию канала между компьютером пользователя и сервером, то большую часть времени канал будет простаивать. В то же время коммутационные возможности сети будут закреплены за данной парой абонентов и будут недоступны другим пользователям сети.

При коммутации *пакетов* все передаваемые пользователем сообщения разбиваются в исходном узле на сравнительно небольшие части, называемые *пакетами*. Напомним, что сообщением называется логически завершенная порция данных — *запрос* на передачу файла, ответ на этот *запрос*, содержащий весь *файл* и т.д. Сообщения могут иметь произвольную длину, от нескольких *байт* до многих *мегабайт*. Напротив, *пакеты* обычно тоже могут иметь переменную длину, но в узких пределах, например от 46 до 1500 *байт*. Каждый *пакет* снабжается *заголовком*, в котором указывается *адресная информация*, необходимая для доставки *пакета* на узел назначения, а также номер *пакета*, который будет использоваться узлом назначения для сборки сообщения (рис. 6.3). *Пакеты* транспортируются *по сети* как независимые информационные блоки. *Коммутаторы* сети принимают *пакеты* от конечных узлов и на основании *адресной информации* передают их друг другу, а в конечном итоге — узлу назначения.

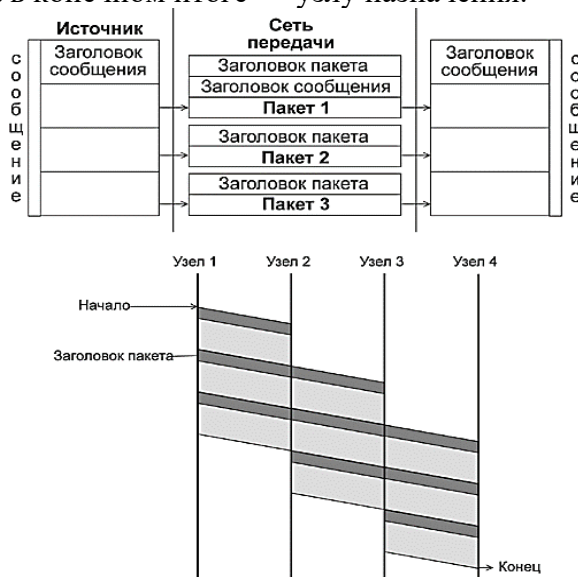


Рис. 6.3. Разбиение сообщения на пакеты

Коммутаторы пакетной сети отличаются от *коммутаторов каналов* тем, что они имеют внутреннюю *буферную память* для временного хранения *пакетов*, если выходной *порт коммутатора* в момент принятия *пакета* занят передачей другого *пакета* (рис. 6.3). В этом случае *пакет* находится некоторое время в очереди *пакетов* в буферной памяти выходного порта, а когда до него дойдет *очередь*, он передается следующему *коммутатору*. Такая схема передачи данных позволяет сглаживать *пульсацию трафика* на магистральных связях между *коммутаторами* и тем самым наиболее эффективно использовать их для повышения пропускной способности сети в целом.

Действительно, для пары абонентов наиболее эффективным было бы предоставление им в единоличное пользование скомутированного канала *связи*, как это делается в сетях с коммутацией каналов. В таком случае время взаимодействия этой пары абонентов было бы минимальным, так как данные без задержек передавались бы от одного абонента другому. Простой канала во время пауз передачи абонентов не интересуют, для них важно быстрее решить свою задачу. *Сеть с коммутацией пакетов* замедляет процесс взаимодействия конкретной пары абонентов, так как их

пакеты могут ожидать в *коммутаторах*, пока по магистральным связям передаются другие *пакеты*, пришедшие в *коммутатор* ранее.

Тем не менее, общий объем передаваемых сетью компьютерных данных в единицу времени при технике коммутации *пакетов* будет выше, чем при технике *коммутации каналов*. Это происходит потому, что пульсации отдельных абонентов в соответствии с законом больших чисел распределяются во времени так, что их пики не совпадают. Поэтому *коммутаторы* постоянно и достаточно равномерно загружены работой, если число обслуживаемых ими абонентов действительно велико. На рис. 6.4 показано, что трафик, поступающий от конечных узлов на *коммутаторы*, распределен во времени очень неравномерно. Однако *коммутаторы* более высокого уровня иерархии, которые обслуживают *соединения* между *коммутаторами* нижнего уровня, загружены более равномерно, и *поток пакетов* в магистральных каналах, соединяющих *коммутаторы* верхнего уровня, имеет почти максимальный коэффициент использования. *Буферизация* сглаживает пульсации, поэтому коэффициент пульсации на магистральных каналах гораздо ниже, чем на каналах абонентского доступа — он может быть равным 1:10 или даже 1:2.

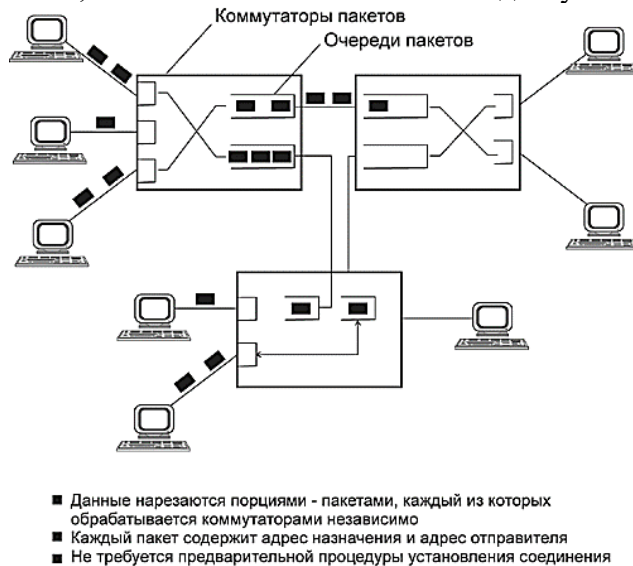


Рис. 6.4. Сглаживание пульсаций трафика в сети с коммутацией пакетов

Более высокая эффективность *сетей с коммутацией пакетов* по сравнению с *сетями с коммутацией каналов* (при равной пропускной способности каналов связи) была доказана в 60-е годы как экспериментально, так и с помощью *имитационного моделирования*. Здесь уместна аналогия с мультипрограммными операционными системами. Каждая отдельная *программа* в такой системе выполняется дольше, чем в однопрограммной системе, когда программе выделяется все *процессорное время*, пока ее выполнение не завершится. Однако общее число программ, выполняемых за единицу времени, в мультипрограммной системе больше, чем в однопрограммной.

Сеть с коммутацией пакетов замедляет процесс взаимодействия конкретной пары абонентов, но повышает пропускную способность сети в целом.

Задержки в источнике передачи:

- время на передачу *заголовков* ;
- **задержки**, вызванные интервалами между передачей каждого следующего *пакета*.

Задержки в каждом *коммутаторе*:

- время *буферизации пакета* ;
- время коммутации, которое складывается из:
 - времени ожидания *пакета* в очереди (переменная величина);
 - времени перемещения *пакета* в выходной порт.

Достоинства коммутации пакетов

1. Высокая общая пропускная способность сети при передаче *пульсирующего трафика*.
2. Возможность динамически перераспределять пропускную способность физических каналов связи между абонентами в соответствии с реальными потребностями их трафика.

Недостатки коммутации пакетов

1. Неопределенность скорости передачи данных между абонентами сети, обусловленная тем, что задержки в очередях буферов *коммутаторов* сети зависят от общей загрузки сети.
2. Переменная величина задержки *пакетов* данных, которая может быть достаточно продолжительной в моменты мгновенных перегрузок сети.
3. Возможные потери данных из-за переполнения буферов.

В настоящее время активно разрабатываются и внедряются методы, позволяющие преодолеть указанные недостатки, которые особенно остро проявляются для чувствительного к задержкам трафика, требующего при этом постоянной скорости передачи. Такие методы называются методами обеспечения *качества обслуживания* (Quality of Service, QoS).

Сети с коммутацией *пакетов*, в которых реализованы методы обеспечения *качества обслуживания*, позволяют одновременно передавать различные виды трафика, в том числе такие важные как телефонный и компьютерный. Поэтому методы коммутации *пакетов* сегодня считаются наиболее перспективными для построения конвергентной сети, которая обеспечит комплексные качественные услуги для абонентов любого типа. Тем не менее, нельзя сбрасывать со счетов и методы *коммутации каналов*. Сегодня они не только с успехом работают в традиционных телефонных сетях, но и широко применяются для образования высокоскоростных постоянных соединений в так называемых первичных (опорных) сетях технологий *SDH* и *DWDM*, которые используются для создания магистральных физических каналов между *коммутаторами* телефонных или компьютерных сетей. В будущем вполне возможно появление новых технологий коммутации, в том или ином виде комбинирующих принципы коммутации *пакетов* и каналов.

Коммутация сообщений

Коммутация сообщений по своим принципам близка к коммутации *пакетов*. Под коммутацией сообщений понимается передача единого блока данных между транзитными компьютерами сети с временной *буферизацией* этого блока на диске каждого компьютера. Сообщение в отличие от *пакета* имеет произвольную длину, которая определяется не технологическими соображениями, а содержанием информации, составляющей сообщение.

Транзитные компьютеры могут соединяться между собой как сетью с коммутацией *пакетов*, так и сетью с коммутацией каналов. Сообщение (это может быть, например, текстовый документ, *файл* с кодом программы, электронное *письмо*) хранится в транзитном компьютере на диске, причем довольно продолжительное время, если *компьютер* занят другой работой или *сеть* временно перегружена.

По такой схеме обычно передаются сообщения, не требующие немедленного ответа, чаще всего сообщения электронной почты. Режим передачи с промежуточным хранением на диске называется режимом "хранения-и-передачи" (*store-and-forward*).

Режим коммутации сообщений разгружает *сеть* для передачи трафика, требующего быстрого ответа, например трафика службы *WWW* или файловой службы.

Количество транзитных компьютеров обычно стараются уменьшить. Если компьютеры подключены к сети с коммутацией *пакетов*, то число промежуточных компьютеров уменьшается до двух. Например, *пользователь* передает почтовое сообщение своему серверу исходящей почты, а тот сразу старается передать его серверу входящей почты *адресата*. Но если компьютеры связаны между собой телефонной сетью, то часто используется несколько промежуточных серверов, так как *прямой доступ* к конечному серверу может быть в данный момент невозможен из-за перегрузки телефонной сети (*абонент* занят) или экономически невыгоден из-за высоких тарифов на дальнюю телефонную связь.

Техника коммутации сообщений появилась в компьютерных сетях раньше техники коммутации *пакетов*, но потом была вытеснена последней, как более эффективной по критерию пропускной способности сети. *Запись* сообщения на *диск* занимает достаточно много времени, и кроме того, наличие дисков предполагает использование в качестве *коммутаторов* специализированных компьютеров, что влечет за собой существенные *затраты* на организацию сети.

Сегодня *коммутация* сообщений работает только для некоторых не оперативных служб, причем чаще всего поверх сети с коммутацией *пакетов*, как служба прикладного уровня.

Маршрутизация

Ключевым методом, позволяющим компьютерам, подключенным к разным сетям обмениваться информацией, является маршрутизация. Пакеты, отправляемые внутри локальной сети, принимаются всеми компьютерами, но каждый обрабатывает лишь те, в которых находит свой адрес. Частью адреса является еще и номер сети, который тоже анализируется каждым получателем. Этот номер должен совпадать с заранее настроенным номером, хранящимся в памяти компьютера. Однако среди компьютеров есть такие, которые подключены одновременно к более чем одной сети. Они называются маршрутизаторами (в англоязычной традиции роутерами, а также шлюзами). Если роутер обнаруживает, что пакет предназначен компьютеру чужой по отношению к отправителю сети, он отправляет его во внешнюю сеть. Соседняя сеть также может передать пакет дальше, пока через цепочку шлюзов он не достигает адресата или не вернется с пометкой, что доставка невозможна.

Маршрутизация служит для приема пакета от одного устройства и передачи его по сети другому устройству через другие сети. Если в сети нет маршрутизаторов, то не поддерживается маршрутизация. Маршрутизаторы направляют (перенаправляют) трафик во все сети, составляющие объединенную сеть.

Для маршрутизации пакета маршрутизатор должен владеть следующей информацией:

- Адрес назначения
- Соседний маршрутизатор, от которого он может узнать об удаленных сетях
- Доступные пути ко всем удаленным сетям
- Наилучший путь к каждой удаленной сети
- Методы обслуживания и проверки информации о маршрутизации

Маршрутизатор узнает об удаленных сетях от соседних маршрутизаторов или от сетевого администратора. Затем маршрутизатор строит таблицу маршрутизации, которая описывает, как найти удаленные сети.

Если сеть подключена непосредственно к маршрутизатору, он уже знает, как направить пакет в эту сеть. Если же сеть не подключена напрямую, маршрутизатор должен узнать (изучить) пути доступа к удаленной сети с помощью статической маршрутизации (ввод администратором вручную местоположения всех сетей в таблицу маршрутизации) или с помощью динамической маршрутизации.

Динамическая маршрутизация — это процесс протокола маршрутизации, определяющий взаимодействие устройства с соседними маршрутизаторами. Маршрутизатор будет обновлять сведения о каждой изученной им сети. Если в сети произойдет изменение, протокол динамической маршрутизации автоматически информирует об изменении все маршрутизаторы. Если же используется статическая маршрутизация, обновить таблицы маршрутизации на всех устройствах придется системному администратору.

Маршрутизатором, или **шлюзом**, называется узел сети с несколькими IP-интерфейсами (содержащими свой MAC-адрес и IP-адрес), подключенными к разным IP-сетям, осуществляющий на основе решения задачи маршрутизации перенаправление дейтаграмм из одной сети в другую для доставки от отправителя к получателю.

Маршрутизаторы представляют собой либо специализированные вычислительные машины, либо компьютеры с несколькими IP-интерфейсами, работа которых управляется специальным программным обеспечением.

Базовые средства передачи данных

К моменту создания первых ИТС наибольшее распространение получили следующие средства связи общего пользования:

- телефонные сети, включающие в себя автоматические телефонные станции, абонентские и соединительные линии, узловые и междугородные телефонные станции;
- телеграфные сети, включающие в себя центры коммутации каналов и сообщений, абонентские и соединительные линии и др.;

- интегрированные системы связи — широкополосные линии связи для телефонии, телеграфии, передачи изображений и телевизионных программ и др.

В настоящее время интенсивно разрабатываются и внедряются *широкополосные цифровые интегрированные системы связи*, предназначенные для телефонных разговоров, передачи изображений и телевизионных программ, а также передачи данных. В интегрированных сетях аналоговые сигналы, в частности речь, передаются в цифровой форме — в виде последовательности байтов. Интегрированные сети реализуют все виды услуг, необходимых в сфере обработки данных, и обеспечивают высокое качество передачи данных, поэтому они являются основой создания современных ИТС.

Локальные вычислительные сети (ЛВС)

Для объединения близко расположенных компьютеров без использования телефонных линий применяются ЛВС.

В свою очередь уже объединенные в ЛВС компьютеры могут быть подключены к Интернету.

Компьютерные сети можно классифицировать по многим признакам, например по удаленности сетевых узлов.

Расстояние между узлами	Масштаб сети	Обозначение
0,1 км	Здание	Локальная сеть (Local Area Network)
1-10 км	Город	Городская сеть (Metropolitan Area Network)
10-1000 км	Страна	Глобальная сеть (Wide Area Network)
Более 1000 км	Планета	Internet

Локальная вычислительная сеть (ЛВС) — это совокупность компьютеров и других средств вычислительной техники (активного сетевого оборудования, принтеров, сканеров и т.п.), объединенных с помощью кабелей и сетевых адаптеров и работающих под управлением сетевой операционной системы. Вычислительные сети создаются для того, чтобы группа пользователей могла совместно задействовать одни и те же ресурсы: файлы, принтеры, модемы, процессоры и т.п. Каждый компьютер в сети оснащается сетевым адаптером, адаптеры соединяются с помощью сетевых кабелей и тем самым связывают компьютеры в единую сеть. Компьютер, подключенный к вычислительной сети, называется рабочей станцией или сервером, в зависимости от выполняемых им функций. Эффективно эксплуатировать мощности ЛВС позволяет применение технологии «клиент/сервер». В этом случае приложение делится на две части: клиентскую и серверную. Один или несколько наиболее мощных компьютеров сети конфигурируются как серверы приложений: на них выполняются серверные части приложений. Клиентские части выполняются на рабочих станциях; именно на рабочих станциях формируются запросы к серверам приложений и обрабатываются полученные результаты.

Структура и принципы строения ЛВС

Различают три наиболее распространенные сетевые архитектуры, которые используют и для одноранговых сетей и для сетей с выделенным файл-сервером. Это так называемые шинная, кольцевая и звездообразная структуры.

В случае реализации шинной структуры все компьютеры связываются в цепочку. Причем на ее концах надо разместить так называемые терминаторы, служащие для гашения сигнала. Если же хотя бы один из компьютеров сети с шинной структурой оказывается неисправным, вся сеть в целом становится неработоспособной. В сетях с шинной архитектурой для объединения компьютеров используется тонкий и толстый кабель. Максимальная теоретически возможная пропускная способность таких сетей составляет 10 Мбит/с. Такой пропускной способности для современных приложений, использующих видео- и мультимедийные данные, явно недостаточно. Поэтому почти повсеместно применяются сети со звездообразной архитектурой.

Для построения сети с звездообразной архитектурой в центре сети необходимо разместить концентратор. Его основная функция — обеспечение связи между компьютерами, входящими в сеть.

То есть все компьютеры, включая файл-сервер, не связываются непосредственно друг с другом, а присоединяются к концентратору. Такая структура надежнее, поскольку в случае выхода из строя одной из рабочих станций все остальные сохраняют работоспособность. В сетях же с шинной топологией в случае повреждения кабеля хотя бы в одном месте происходит разрыв единственного физического канала, необходимого для движения сигнала. Кроме того, сети со звездообразной топологией поддерживают технологии Fast Ethernet и Gigabit Ethernet, что позволяет увеличить пропускную способность сети в десятки и даже сотни раз (разумеется, при использовании соответствующих сетевых адаптеров и кабелей).

Кольцевая структура используется в основном в сетях Token Ring и мало чем отличается от шинной. Также в случае неисправности одного из сегментов сети вся сеть выходит из строя. Правда, отпадает необходимость в использовании терминаторов.

В сети любой структуры в каждый момент времени обмен данными может происходить только между двумя компьютерами одного сегмента. В случае ЛВС с выделенным файл-сервером – это файл-сервер и произвольная рабочая станция; в случае одноранговой ЛВС – это любые две рабочие станции, одна из которых выполняет функции файл-сервера. Упрощенно диалог между файл-сервером и рабочей станцией выглядит так: открыть файл – подтвердить открытие файла; передать данные файла – пересылка данных; закрыть файл – подтверждение закрытия файла. Управляет диалогом сетевая операционная система, клиентские части которой должны быть установлены на рабочих станциях.

Конфигурация связей

При объединении в сеть большего числа компьютеров возникает целый комплекс новых проблем.

Топология физических связей

При объединении в сеть большего числа компьютеров возникает целый комплекс новых проблем.

В первую очередь необходимо выбрать способ организации физических связей, то есть *топологию*. Под топологией вычислительной сети понимается конфигурация графа, вершинам которого соответствуют компьютеры сети (иногда и другое оборудование, например концентраторы), а ребрам – физические связи между ними. Компьютеры, подключенные к сети, часто называют *станциями* или *узлами* сети.

Заметим, что конфигурация *физических связей* определяется электрическими соединениями компьютеров между собой и может отличаться от конфигурации *логических связей* между узлами сети. Логические связи представляют собой маршруты передачи данных между узлами сети и образуются путем соответствующей настройки коммуникационного оборудования.

Выбор топологии электрических связей существенно влияет на многие характеристики сети. Например, наличие резервных связей повышает надежность сети и делает возможным балансирование загрузки отдельных каналов. Простота присоединения новых узлов, свойственная некоторым топологиям, делает сеть легко расширяемой. Экономические соображения часто приводят к выбору топологий, для которых характерна минимальная суммарная длина линий связи. Рассмотрим некоторые, наиболее часто встречающиеся топологии.

Полносвязная топология (рис. 1.10, а) соответствует сети, в которой каждый компьютер сети связан со всеми остальными. Несмотря на логическую простоту, этот вариант оказывается громоздким и неэффективным. Действительно, каждый компьютер в сети должен иметь большое количество коммуникационных портов, достаточное для связи с каждым из остальных компьютеров сети. Для каждой пары компьютеров должна быть выделена отдельная электрическая линия связи. Полносвязные топологии применяются редко, так как не удовлетворяют ни одному из приведенных выше требований. Чаще этот вид топологии используется в многомашинных комплексах или глобальных сетях при небольшом количестве компьютеров.

Все другие варианты основаны на неполносвязных топологиях, когда для обмена данными между двумя компьютерами может потребоваться промежуточная передача данных через другие узлы сети.

Ячеистая топология (*mesh*) получается из полностью связной путем удаления некоторых

возможных связей (рис. 1.10, б). В сети с ячеистой топологией непосредственно связываются только те компьютеры, между которыми происходит интенсивный обмен данными, а для обмена данными между компьютерами, не соединенными прямыми связями, используются транзитные передачи через промежуточные узлы. Ячеистая топология допускает соединение большого количества компьютеров и характерна, как правило, для глобальных сетей.

Общая шина (рис. 1.10, в) является очень распространенной (а до недавнего времени самой распространенной) топологией для локальных сетей. В этом случае компьютеры подключаются к одному коаксиальному кабелю по схеме «монтажного ИЛИ». Передаваемая информация может распространяться в обе стороны. Применение общей шины снижает стоимость проводки, унифицирует подключение различных модулей, обеспечивает возможность почти мгновенного широковещательного обращения ко всем станциям сети. Таким образом, основными преимуществами такой схемы являются дешевизна и простота разводки кабеля по помещениям. Самый серьезный недостаток общей шины заключается в ее низкой надежности: любой дефект кабеля или какого-нибудь из многочисленных разъемов полностью парализует всю сеть. К сожалению, дефект коаксиального разъема редкостью не является. Другим недостатком общей шины является ее невысокая производительность, так как при таком способе подключения в каждый момент времени только один компьютер может передавать данные в сеть. Поэтому пропускная способность канала связи всегда делится здесь между всеми узлами сети.

Топология *звезда* (рис. 1.10, г). В этом случае каждый компьютер подключается отдельным кабелем к общему устройству, называемому *концентратором*, который находится в центре сети. В функции концентратора входит направление передаваемой компьютером информации одному или всем остальным компьютерам сети. Главное преимущество этой топологии перед общей шиной - существенно большая надежность. Любые неприятности с кабелем касаются лишь того компьютера, к которому этот кабель присоединен, и только неисправность концентратора может вывести из строя всю сеть. Кроме того, концентратор может играть роль интеллектуального фильтра информации, поступающей от узлов в сеть, и при необходимости блокировать запрещенные администратором передачи.

К недостаткам топологии типа звезда относится более высокая стоимость сетевого оборудования из-за необходимости приобретения концентратора. Кроме того, возможности по наращиванию количества узлов в сети ограничиваются количеством портов концентратора. Иногда имеет смысл строить сеть с использованием нескольких концентраторов, иерархически соединенных между собой связями типа звезда (рис. 1.10, д). В настоящее время иерархическая звезда является самым распространенным типом топологии связей как в локальных, так и глобальных сетях.

В сетях с *кольцевой* конфигурацией (рис. 1.10, е) данные передаются по кольцу от одного компьютера к другому, как правило, в одном направлении. Если компьютер распознает данные как «свои», то он копирует их себе во внутренний буфер. В сети с кольцевой топологией необходимо принимать специальные меры, чтобы в случае выхода из строя или отключения какой-либо станции не прервался канал связи между остальными станциями. Кольцо представляет собой очень удобную конфигурацию для организации обратной связи - данные, сделав полный оборот, возвращаются к узлу-источнику. Поэтому этот узел может контролировать процесс доставки данных адресату. Часто это свойство кольца используется для тестирования связности сети и поиска узла, работающего некорректно. Для этого в сеть посылаются специальные тестовые сообщения.

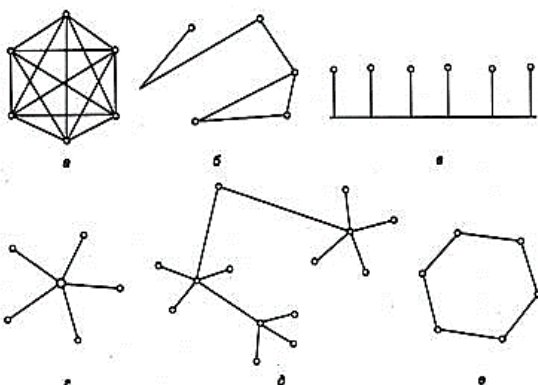


Рис. 1.10 . Типовые топологии сетей

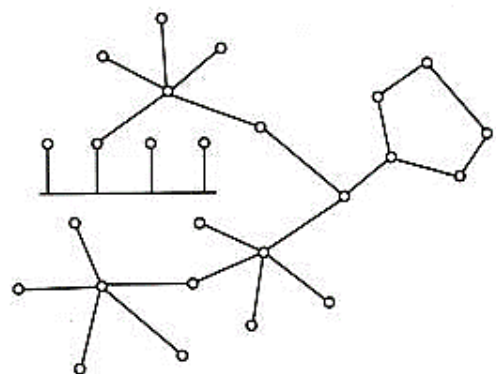


Рис. 1.11 . Смешанная топология

В то время как небольшие сети, как правило, имеют типовую топологию - звезда, кольцо или общая шина, для крупных сетей характерно наличие произвольных связей между компьютерами. В таких сетях можно выделить отдельные произвольно связанные фрагменты (подсети), имеющие типовую топологию, поэтому их называют сетями со *смешанной топологией* (рис. 1.11)

Стандарты, соглашения и рекомендации

Работы по стандартизации вычислительных сетей ведутся большим количеством организаций. В зависимости от статуса организаций различают следующие виды **стандартов**:

- стандарты отдельных фирм;
- стандарты специальных комитетов и объединений, создаваемых несколькими фирмами;
- национальные стандарты;
- международные стандарты, например, модель и стек коммуникационных протоколов, сети frame relay, ISDN, модемы и многие другие.

Некоторые стандарты, непрерывно развиваясь, могут переходить из одной категории в другую. Более того, ввиду широкого распространения некоторые фирменные стандарты становятся основой для национальных и международных стандартов де-юре. Например, стандарт Ethernet.

При передаче сообщений оба участника сетевого обмена должны принять множество **соглашений**. Например, они должны согласовать уровни и форму электрических сигналов, способ определения длины сообщений, договориться о методах контроля достоверности и т. п. Другими словами, соглашения должны быть приняты для всех уровней, начиная от самого низкого - уровня передачи битов - до самого высокого, реализующего сервис для пользователей сети.

Формализованные правила, определяющие последовательность и формат сообщений, которыми обмениваются сетевые компоненты, лежащие на одном уровне, но в разных узлах, называются *протоколом*.

Из того, что *протокол является соглашением*, принятым двумя взаимодействующими объектами, в данном случае двумя работающими в сети компьютерами, совсем не следует, что он обязательно является стандартным. Но на практике при реализации сетей стремятся использовать стандартные протоколы.

В начале 80-х годов ряд международных организаций по стандартизации - ISO, ITU-T и некоторые другие - разработали модель, которая сыграла значительную роль в развитии сетей. Эта модель называется *моделью взаимодействия открытых систем (Open System Interconnection, OSI)* или моделью OSI. Модель OSI определяет различные уровни взаимодействия систем, дает им стандартные имена и указывает, какие функции должен выполнять каждый уровень. Модель разбиения на уровни OSI является **рекомендуемой** моделью.

В модели OSI (рис. 1.25) средства взаимодействия делятся на семь уровней: прикладной, представительный, сеансовый, транспортный, сетевой, канальный и физический. Каждый уровень имеет дело с одним определенным аспектом взаимодействия сетевых устройств.

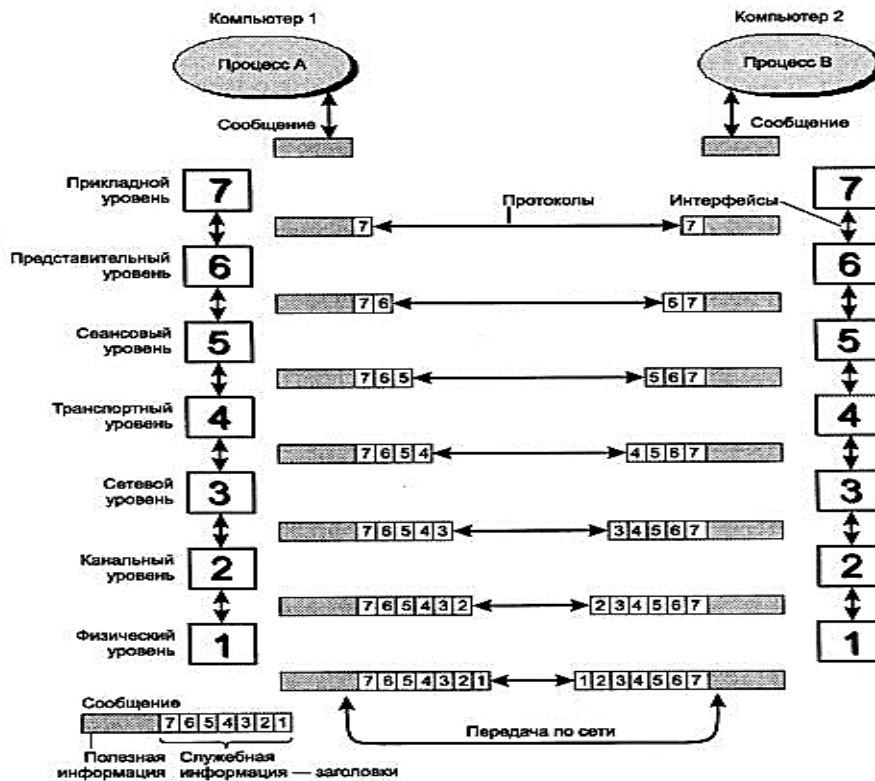


Рис. 1.25 . Модель взаимодействия открытых систем ISO/OSI

Модель OSI описывает только системные средства взаимодействия, реализуемые операционной системой, системными утилитами, системными аппаратными средствами. Модель не включает средства взаимодействия приложений конечных пользователей. Свои собственные протоколы взаимодействия приложения реализуют, обращаясь к системным средствам. Поэтому необходимо различать уровень взаимодействия приложений и прикладной уровень.

Итак, пусть приложение обращается с запросом к прикладному уровню, например к файловой службе. На основании этого запроса программное обеспечение прикладного уровня формирует сообщение стандартного формата. Обычное сообщение состоит из заголовка и поля данных. Заголовок содержит служебную информацию, которую необходимо передать через сеть прикладному уровню машины-адресата, чтобы сообщить ему, какую работу надо выполнить. В нашем случае заголовок, очевидно, должен содержать информацию о месте нахождения файла и о типе операции, которую необходимо над ним выполнить. Поле данных сообщения может быть пустым или содержать какие-либо данные, например те, которые необходимо записать в удаленный файл. Но для того чтобы доставить эту информацию по назначению, предстоит решить еще много задач, ответственность за которые несут нижележащие уровни.

После формирования сообщения прикладной уровень направляет его вниз по стеку представителю уровня. Протокол представительного уровня на основании информации, полученной из заголовка прикладного уровня, выполняет требуемые действия и добавляет к сообщению собственную служебную информацию - заголовок представительного уровня, в котором содержатся указания для протокола представительного уровня машины-адресата. Полученное в результате сообщение передается вниз сеансовому уровню, который в свою очередь добавляет свой заголовок, и т. д. (Некоторые реализации протоколов помещают служебную информацию не только в начале сообщения в виде заголовка, но и в конце, в виде так называемого «концевика».) Наконец, сообщение достигает нижнего, физического уровня, который собственно и передает его по линиям связи машине-адресату. К этому моменту сообщение «обрастает» заголовками всех уровней (рис. 1.26).

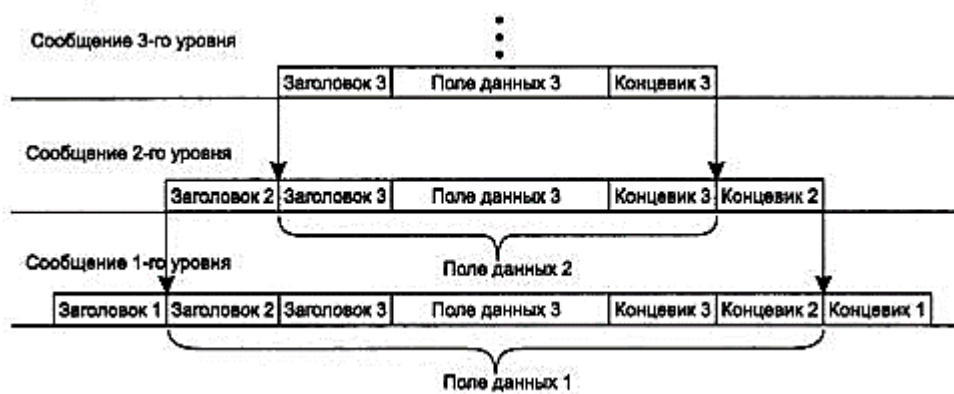


Рис. 1.26 . Вложенность сообщений различных уровней

Когда сообщение по сети поступает на машину - адресат, оно принимается ее физическим уровнем и последовательно перемещается вверх с уровня на уровень. Каждый уровень анализирует и обрабатывает заголовок своего уровня, выполняя соответствующие данному уровню функции, а затем удаляет этот заголовок и передает сообщение вышележащему уровню.

Программное обеспечение компьютерных сетей

Программное обеспечение компьютерных сетей обеспечивает организацию коллективного доступа к вычислительным и информационным ресурсам сети, динамическое распределение и перераспределение ресурсов сети с целью повышения оперативности обработки информации и максимальной загрузки аппаратных средств, а также в случае отказа и выхода из строя отдельных технических средств и т.д.

Программное обеспечение вычислительных сетей включает три компонента:

- общее программное обеспечение, образуемое базовым ПО отдельных ЭВМ, входящих в состав сети;
- специальное программное обеспечение, образованное прикладными программными средствами, отражающими специфику предметной области пользователей при реализации задач управления;
- системное сетевое программное обеспечение, представляющее комплекс программных средств, поддерживающих и координирующих взаимодействие всех ресурсов вычислительной сети как единой системы.

Особая роль в ПО вычислительной сети отводится системному сетевому программному обеспечению, функции которого реализуются в виде распределенной операционной системы сети.

Операционная система сети включает в себя набор управляющих и обслуживающих программ, обеспечивающих:

- межпрограммный метод доступа (возможность организации связи между отдельными прикладными программами комплекса, реализуемыми в различных узлах сети);
- доступ отдельных прикладных программ к ресурсам сети (и в первую очередь к устройствам ввода-вывода);
- синхронизацию работы прикладных программных средств в условиях их обращения к одному и тому же вычислительному ресурсу;
- обмен информацией между программами с использованием сетевых «почтовых ящиков»;
- выполнение команд оператора с терминала, подключенного к одному из узлов сети, на каком-либо устройстве, подключенном к другому удаленному узлу вычислительной сети;
- удаленный ввод заданий, вводимых с любого терминала, и их выполнение на любой ЭВМ в пакетном или оперативном режиме;
- обмен наборами данных (файлами) между ЭВМ сети;
- доступ к файлам, хранимым в удаленных ЭВМ, и обработку этих файлов;
- защиту данных и вычислительных ресурсов сети от несанкционированного доступа;
- выдачу различного рода справок об использовании информационных, программных и технических ресурсов сети;

- передачу текстовых сообщений с одного терминала пользователя на другие (электронная почта).

С помощью операционной системы сети:

- устанавливается последовательность решения задач пользователя;
- задачи пользователя обеспечиваются необходимыми данными, хранящимися в различных узлах сети;
- контролируется работоспособность аппаратных и программных средств сети;
- обеспечивается плановое и оперативное распределение ресурсов в зависимости от возникающих потребностей различных пользователей вычислительной сети.

Выполняемое с помощью операционной системы сети управление включает: планирование сроков и очередности получения и выдачи информации абонентам; распределение решаемых задач по ЭВМ сети; присвоение приоритетов задачам и выходным сообщениям; изменение конфигурации сети ЭВМ; распределение информационных вычислительных ресурсов сети для решения задач пользователя.

Оперативное управление процессом обработки информации с помощью операционной системы сети помогает организовать: учет выполнения заданий (либо определить причины их невыполнения); выдачу справок о прохождении задач в сети; сбор данных о работах, выполняемых в сети.

ОС отдельных ЭВМ, входящих в состав вычислительной сети, поддерживают потребности пользователей во всех традиционных видах обслуживания: средствах автоматизации программирования и отладки, доступа к пакетам прикладных программ и информации локальных баз данных и т.д.

Назначение и основные функции операционных систем

Назначение и основные функции операционных систем (ОС)

Операционная система (ОС) - это комплекс программного обеспечения, предназначенный для снижения стоимости программирования, упрощения доступа к системе, повышения эффективности работы.

Цель создания операционной системы - получить экономический выигрыш при использовании системы, путем увеличения производительности труда программистов и эффективности работы оборудования.

Функции операционной системы:

- связь с пользователем в реальном времени для подготовки устройств к работе, переопределение конфигурации и изменения состояния системы.
- выполнение операций ввода-вывода; в частности, в состав операционной системы входят программы обработки прерываний от устройств ввода-вывода, обработки запросов к устройствам ввода-вывода и распределения этих запросов между устройствами.
- управление памятью, связанное с распределением оперативной памяти между прикладными программами.
- управление файлами; основными задачами при этом являются обеспечение защиты, управление выборкой и сохранение секретности хранимой информации.
- обработка исключительных условий во время выполнения задачи
- появление арифметической или машинной ошибки, прерываний, связанных с неправильной адресацией или выполнением привилегированных команд.
- вспомогательные, обеспечивающие организацию сетей, использование служебных программ и языков высокого уровня.

Основные характеристики операционной системы UNIX

Операционная система UNIX разработана в 70-х годах Кеном Томпсоном и Деннисом Ритчи в Bell Laboratory и первоначально предназначалась для проведения исследовательских работ. Однако концептуальная целостность системы и целый ряд новых нетрадиционных и прогрессивных решений, заложенных в нее при создании, показали преимущества UNIX по сравнению с другими операционными системами этого класса. Система UNIX быстро распространилась и сейчас активно используется на многих вычислительных установках. В нашей стране аналогом операционной системы UNIX является ИНМОС. Сначала эта система была ориентирована на СМ ЭВМ, а затем перенесена и в сферу ЕС ЭВМ. К несомненным достоинствам UNIX следует отнести:

- концептуальное единство
- простоту
- инструментальность
- мобильность
- эффективность

Концептуальное единство заключается в том, что система UNIX "обозрима", и пользователь, работающий с ней, как правило, не только использует ее возможности, но и хорошо представляет, как они реализуются.

Простота системы проистекает из того условия, что средства системы выразительны и имеют удобную мнемонику. Допускается изучение по частям, что дает возможность пользователю ограничиться изучением только того набора средств, который ему в данный момент необходим.

Инструментальность проявляется в насыщении различными инструментальными средствами, облегчающими процесс конструирования программного обеспечения. Их можно разделить на следующие группы:

- утилиты различного назначения
- средства работы с текстами
- средства поддержки разработок программного обеспечения
- средства генерации программ анализа
- интерпретатор shell, являющийся важным инструментальным средством, позволяющий на

уровне командного языка строить новые инструментальные средства проблемной направленности.

Мобильность означает возможность переноса системы с одной машинной архитектуры на другую с минимальными затратами.

Эффективность заключается в конструктивных особенностях, реализованных в системе, освобождающих пользователя от решения многих проблем, свойственных другим системам, и значительно повышают продуктивность практического использования системы для решения самых разнообразных задач.

Недостатки системы:

- не поддерживается режим реального времени
- слабая устойчивость к аппаратным сбоям
- снижение эффективности при решении однотипных задач
- слабо развиты средства взаимодействия и синхронизации процессов

Основные характеристики операционной системы MS-DOS

Операционная система MS-DOS (дисктовая операционная система фирмы Microsoft), была разработана в 1981 г. Билом Гейтсом - президентом фирмы Microsoft, одновременно с машинами типа IBM PC и стала для них доминирующей. К настоящему времени разработано несколько версий системы. MS-DOS во многом напоминает по своим возможностям ОС UNIX. Предоставляемые MS DOS возможности обеспечивают, с одной стороны, удобный доступ к имеющимся прикладным пакетам и программам для непрофессиональных пользователей, с другой стороны, создают хорошую среду для разработки программного обеспечения. MS DOS является стандартом для 16-разрядных микро ЭВМ. В отличие от CP/M MS-DOS обеспечивает организацию многоуровневых каталогов, имеет более развитый командный язык.

Структура MS-DOS.

Операционная система MS-DOS состоит из трех основных подсистем:

- модуль взаимодействия с базовой системой ввода-вывода (файл IO.SYS)
- собственно операционная система, обеспечивающая взаимодействие с программами пользователя. Она состоит из программы файловой системы, программ блочного обмена с дисками и других встроенных операций доступных программ пользователей (MSDOS.SYS)

- командный процессор (файл command.com) Все подсистемы должны располагаться на диске, с которого происходит загрузка операционной системы. При запуске системы (любая операция перезагрузки, либо при включении питания) в память считывается и получает управление специальная программа - так называемая программа начальной загрузки. Она помещается на все дискеты, чтобы напечатать сообщение об ошибке при попытке запустить систему с дискетой в формате отличном от формата MS-DOS

При получении управления программой начальной загрузки просматривается оглавление диска - проверяется, что первые два файла - это IO.SYS и MSDOS.SYS. Если они не обнаружены, на экран выдается сообщение об ошибке, если обнаружены, оба файла считываются в оперативную память и управление передается в модуль взаимодействия с базовой системой ввода-вывода (IO.SYS).

Подпрограмма инициализации (начала работы) в IO.SYS определяет состояние оборудования, приводит в действие дисковую систему и подключенные устройства, загружает драйверы устройств и устанавливает значения специальных управляющих блоков, связанных с обработкой прерываний. Затем она выполняет настройку адресов в ядре MS DOS и передает ему управление. Ядро MS DOS инициализирует свои внутренние рабочие таблицы, создает управляющие таблицы и возвращает управление модулю взаимодействия с BIOS. Последнее действие IO.SYS - загрузка командного процессора по адресу, установленному подпрограммой инициализации ядра MS DOS. Затем управление передается COMMAND. COM. Модуль взаимодействия с базовой системой ввода/вывода реализует набор операций работы с дисками и устройствами ввода/вывода. Только эта часть MS DOS непосредственно взаимодействует с внешними устройствами, только она зависит от особенностей и характеристик внешних устройств, используемых в конкретных компьютерах. В ней определена логика взаимодействия с устройствами ввода-вывода адресов подключения, набор команд контроллера дисков и т. д. Все другие компоненты MS DOS общаются с внешним миром только через модуль взаимодействия с BIOS.

Пользователь из своих прикладных программ может обращаться к некоторым MS DOS

командам. Имеются программы для ввода с клавиатуры, для вывода на терминал и на печать, для формирования блоков управления файлами, управления памятью, обработки даты и времени, операций над дисками, каталогами и файлами. Ядро MS DOS реализует все функции, связанные с файловой организацией информации на дисках, управлением дисковыми, распределением пространства и работой с их справочниками. MS DOS размещает (форматирует) все диски и дискеты с размером сектора 512 байт. Область для MS DOS (вся дискета или раздел на твердом диске) распределена следующим образом:

- блок начальной загрузки
- таблица размещения файлов
- копия таблицы размещения файлов
- корневой каталог
- область данных

Файлам выделяется пространство в области данных по мере необходимости, когда происходит фактическая запись; предварительного распределения не производится. Пространство выделяется порциями, называемыми кластерами. На односторонних дискетах кластер равен одному блоку; на двухсторонних каждый кластер состоит из двух блоков. Размер кластера для твердого диска определяется при разметке командой FORMAT и зависит от размера раздела MS DOS.

Таблица размещения файлов (File Allocation Table - FAT) связывает кластеры одного файла в цепочку.

Кластеры устроены так, что минимизированы перемещения головок при работе с многосторонними носителями. Все пространство одной дорожки или одного цилиндра заполняется информацией, после чего происходит переход к следующей дорожке или цилиндру. При этом сначала используются последовательные секторы для головки с наименьшим сектором, после чего секторы следующей головки и так далее до последней головки, затем происходит переход к следующему цилиндру.

Файлы пишутся на диск в область данных обязательно последовательно. Пространство области данных распределяется по одному кластеру, при этом уже занятые кластеры пропускаются.

Выделяется первый найденный свободный кластер вне зависимости от его физического расположения на диске. Это обеспечивает наиболее эффективное использование дискового пространства, так как кластеры, освобожденные при удалении файла, становятся доступными для размещения новых файлов.

Командный процессор организует взаимодействие системы с пользователем на языке команд MS DOS. Он читает команды, введенные с пульта оператора, анализирует их и выполняет либо непосредственно (встроенные команды), либо загрузив в оперативную память программу, соответствующую этой команде (загружаемые команды), и передав ей управление.

Windows 3.1 и Windows 3.11

По сути дела Windows 3.1 и 3.11 является всего-навсего надстройкой над DOS, однако между ними существуют серьезные различия - именно они позволяют называть Windows операционной системой. Графический интерфейс, как оказывается здесь не главное.

Итак, что же можно сказать про Windows? Начнем с того, что в Windows вы можете последовательно запустить несколько (а не строго одно, как в DOS) приложений и переключаться между ними в процессе работы. Некоторые приложения, в зависимости от задачи, могут продолжать работать, находясь в запущенном, но неактивном состоянии.

В Windows используется неактивный режим работы процессора (protected mode), и программа пользователя уже не может влезть в какую ей угодно область памяти и делать там что вздумается.

Большим преимуществом Windows 3.11 стала возможность работы в одно-ранговой сети или с выделенным сервером. Теоретически, можно забыть про приобретение специального сетевого программного обеспечения и обойтись только средствами Windows 3.11. Однако на практике в локальной сети всё-таки лучше ставить специальное сетевое ПО, а уже поверх него - Windows 3.11. Тем более что в Windows предусмотрена поддержка не только своей сети, но и других сетевых протоколов.

Возможность использования в программах виртуальной памяти - (иными словами, выделение программе шести мегабайт памяти на машине с физическими четырьмя) также весьма удобна. И хотя в таком режиме компьютер заметно замедляет свою работу, бывает очень важно, чтобы

“требовательная” программа работала уж как-нибудь, чем никак.

Windows распахивает перед пользователями фантастический мир мультимедиа, измерения которого системе DOS и не снились. DOS могла позволить воспроизведение максимум небольших мультфильмов - способности Windows трудно перечислить: это компьютерные игры, электронные энциклопедии, интерактивная графика и многое другое.

По своим возможностям Windows значительно превосходит DOS, но и требования к аппаратным ресурсам компьютера предъявляет немалые. Так, для работы в защищённом режиме компьютер должен быть оснащён как минимум 386-м процессором, а уж памяти Windows потребляет исходя из принципа “чем больше, тем лучше”. И если DOS на “двушке” с мегабайтом памяти работала быстро и уверенно, то Windows, даже версии 3.0, на такой машине работает крайне медленно.

Сегодня совмещение DOS + Windows на персональных компьютерах встречается наиболее часто; для Windows разработано несметное количество приложений, игр и

Windows 95

Windows 95 выделяется среди других систем по совокупности своих характеристик: удобства, совместимости, функциональных возможностей и быстродействия.

Создавая Windows 95, корпорация Microsoft, по-видимому, задумала такую систему, работать с которой мог бы человек, не окончивший даже среднюю школу. Инсталлировать Windows 95 не сложнее, чем хлеба к обеду нарезать.

Правда преемственность предыдущих версий Windows только этим не ограничивается: Windows 95 имеет много общего со своими предшественницами. Она устанавливается, по сути дела, поверх MS-DOS, и, честно говоря безразлично, что она пишет в ответ на команду “ver” в режиме DOS:MS-DOS version x.xx или Windows 95.

Несмотря на то что Windows 95 разрекламирована как полноценная 32-разрядная операционная система, в действительности же она имеет 16-разрядное ядро. Как это ни прискорбно, для разработки 32-разрядных приложений необходимо запускать специальные утилиты - аналогично тому, как в Windows 3.11 ставился модуль Win32S. (Кстати, это и есть тот же Win32S, только видоизменённый для Windows 95)

Из всех усовершенствований, реализованных в Windows 95, для повышения производительности работы пользователя, вероятно, важнее всего значительные усовершенствования в интерфейсе. Изменения в нём, по сравнению с Windows 3.x в самом деле поразительны, но не меньше бросается в глаза то, как много в нем заимствований из Mac OS и OS/2.

Например:

- при нажатии правой кнопки мыши появляется контекстно-зависимое меню (OS/2).
- корзина “Recicle Bin”, аналог Мусорного Ведро (“Trasch” Mac OS) и т. д.
- программы, документы и ярлыки (указатели на другие файлы могут размещаться на “Рабочем столе” (OS/2)).

Поддержка сетевых протоколов в Windows 95 немного расширилась по сравнению с предыдущими версиями Windows: например, появилась поддержка протокола TCP/IP; по отношению к локальным сетям политика не претерпела изменений. Сохранился свой протокол обмена между компьютерами и возможность поддержки других сетевых протоколов.

Новая ОС не только выполняет подавляющее большинство существующих программ для Windows 3.x и DOS, но и совместима с драйверами реального режима для этих систем. Использование таких драйверов может ослабить устойчивость работы 32-разрядной системы, зато устраняет сложности, возникающие из-за отсутствия нужного драйвера для того или иного периферийного устройства. Эту проблему никак не удаётся решить ни в Windows NT ни в OS/2 Warp.

Требования Windows 95 к аппаратному обеспечению несколько выросли по сравнению с Windows 3.11. В первую очередь они коснулись объёма оперативной памяти, необходимой для нормальной работы.

Windows 95 вышла на рынок сравнительно недавно, однако под неё написано уже много приложений: ведущие производители ПО связывают с ней большие надежды и переводят свои популярные продукты на рельсы Windows 95.

OS/2

Операционная система OS/2, разработанная фирмой IBM, даже в ранних версиях зарекомендовала себя как весьма мощная ОС. Думаю, что не ошибусь, сказав, что OS/2 стала первой реально многозадачной операционной системой на персоналках, к тому же в своей основе она является объектно-ориентированной. В общем-то, складывается такое впечатление, что если Windows разрабатывалась начиная с интерфейса, то OS/2 создавалась, как и положено операционным системам, начиная с ядра. OS/2 является действительно 32-разрядной операционной системой, и ей не требуется никаких дополнений для работы с 32-разрядными приложениями.

Апологеты Windows довольно долго обвиняли OS/2 в том, что она не “понимает” программ, написанных для Windows, - что и говорить, их число огромно, а некоторые из них уникальны. Уже в версии 2.0 программисты IBM исправили свою ошибку и включили в OS/2 сессию Windows. И сделали они это, надо отдать должное, весьма неплохо: Windows-приложения стали работать на порядок быстрее (кстати, DOS приложения под OS/2 тоже работают быстрее, нежели под DOS).

Первая версия OS/2 вообще не имела графического интерфейса (presentation manager) - он появился относительно недавно. Правда, у ранних версий этой ОС и запросы к аппаратному обеспечению сравнительно невысоки. Так что если вам нужна реальная многозадачность, но денег на мощный компьютер не хватает, то ранние версии OS/2 - это неплохой вариант решения данной проблемы. Более поздние версии OS/2 получили графический интерфейс. И хотя интерфейс OS/2 иногда обвиняют в том, что с ним не возможно работать, многие не склонны разделять это мнение; некоторые пользователи, напротив, устанавливают в Windows программу, реализующую интерфейс от OS/2.

Основное достоинство OS/2 - это, несомненно, возможность работать в режиме разделения времени. Она позволяет выполнять вам несколько задач одновременно: например, форматировать дискету и одновременно компилировать программу. При этом выполнение задач почти не замедляется.

Работа с сетями в OS/2 ориентирована в первую очередь на поддержку протокола TCP/IP, однако система неплохо работает и с другими протоколами, например IPX.

Нельзя не упомянуть и о том, что недавно была выпущена локализованная версия OS/2 Warp 3.0. Теперь стало проще разбираться в многочисленных настройках самой OS/2, а так же сеансов DOS и Windows.

Само собой разумеется, что широкие возможности OS/2 обходятся пользователю недёшево. И хотя IBM объявила, что OS/2 Warp работает на 386-м компьютере с четырьмя мегабайтами оперативной памяти, специалисты замечают: если вы хотите, чтобы OS/2 работала без проблем, помножьте все аппаратные требования, которые указаны в руководстве, на два.

В целом же, кто имел раньше дело с Win 95, NT и Mac OS сочли OS/2 наименее удобной в работе среди рассматриваемых ОС. Одним из очень больших недостатков является например то, что Warp позволяет удалять файлы простым перемещением их на пиктограмму Shredder (“Дробилка”), однако по умолчанию она не может восстановить файл, который был стёрт ошибочно. Она не выдаёт предупреждений при попытке удаления жизненно важных системных файлов.

OS/2 Warp 3.0 остаётся привлекательной для тех, кто любит настраивать интерфейс по своему вкусу и кому не требуется очень много готовых прикладных пакетов. Высококачественные средства программирования, имеющиеся в данной ОС, обеспечивают этой системе успех у фирм, разрабатывающих собственное программное обеспечение для внутреннего пользования.

Иными словами, имеет смысл устанавливать в том случае, если вам действительно необходим режим разделения времени между задачами, а также возможности, которые другие операционные системы предоставить не могут.

Windows 98

Windows 98 позиционируется компанией Microsoft как обновление для Windows 95. Сама Microsoft настаивает на том, что, не будучи “революционно новой”, очередная ОС обеспечивает прирост производительности системы и большую стабильность в работе.

Кроме того, в Windows 98 реализован ряд новых возможностей, позволяющих, к примеру, работать с USB-устройствами, DVD-дисковыми и т.д.

В Windows 98 значительно богаче набор средств для диагностики и разрешения конфликтов, чем в Windows 95, включая Version Conflict Manager и Maintenance Wizard. Microsoft предполагает, что благодаря этим средствам уменьшение числа обращений пользователей к службам технической

поддержки.

Добавлена версия Internet Explorer 4.0. Windows 98 улучшает качество воспроизведения графики, звука и мультимедийных приложений, созданных по новейшим технологиям.

Через некоторое время появилась на свет вторая редакция популярной операционной системы Windows 98 (полное название версии 4.10.2222). Windows 98 Second Edition – скорее, сборник апдейтов и мелких дополнений, чем что-то революционно новое.

Во-первых, это новый Explorer версии 5.0.

Во-вторых, в новой редакции системы появился прокси-сервер ICS (Internet Connection Sharing - совместное использование Internet), который позволяет в небольшой локальной сети организовать совместный доступ в Интернет при помощи одного модема.

В-третьих, это новая версия NetMeeting - 3.0. Последняя версия NetMeeting расширяет возможности проведения сетевых конференций, повышает быстродействие и обеспечивает безопасность и поддержку стандартов Интернета.

В четвертых это Service Pack, то есть сборник апдейтов и устранений ошибок предыдущей версии Windows.

В-пятых, в систему введена поддержка IEEE 1394 и ACPI, а также улучшена поддержка USB.

Преимущества:

Простота использования и доступа в Интернет. Динамическая справочная система на основе веб-технологии и 15 программ-мастеров упрощают использование компьютера. Веб-совместимый интерфейс пользователя Windows 98 облегчает поиск, унифицируя представление информации в компьютере, локальной сети и Вебе. Второе издание Windows 98 обеспечивает возможность одновременного доступа в Интернет с нескольких сетевых компьютеров через одно общее подключение.

Высокая производительность и надежность. Сокращение времени запуска приложений, новые средства очистки диска и повышения эффективности его работы. Все это стало возможным благодаря новшествам, превращающим Windows 98 в мощную и надежную операционную систему.

Поддержка аппаратных средств нового поколения. Использование преимуществ новейших стандартов и технологий, таких как шина USB, DVD и IEEE 1394, расширение возможностей за счет подключения к одному компьютеру нескольких мониторов,

Windows NT

С такими операционными системами, как Windows NT, рядовому пользователю приходится сталкиваться довольно редко: разве что где-нибудь на работе или на выставке. Несмотря на то что Windows NT названием и интерфейсом похожа на другие ОС корпорации Microsoft, она значительно от них отличается - Windows NT предназначена в первую очередь для крупных сетей. Windows NT, в отличие от Windows 3.11, является полноценной 32-разрядной операционной системой с широкими возможностями; благодаря развитым сетевым возможностям она может использоваться при интеграции нескольких сетей.

Система не поддерживает идеологию Plug&Play. Однако, по целому ряду причин для некоторых пользователей именно эта ОС является наилучшей. Если характер вашей работы таков, что любая порча данных может обойтись очень дорого, то серьезные меры по обеспечению устойчивости и безопасности, реализованные в Windows NT могут быть спасением. Windows NT - система скорее для корпоративных, чем для домашних пользователей. В этой промышленной версии Windows фирмы Microsoft основной упор сделан на безопасность и надёжность в ущерб всему остальному, в том числе и удобству пользователя.

Windows 2000

В феврале 2000 г. в Москве прошла презентация новой системы Windows 2000. Платформа Windows 2000 представляет собой операционную систему нового поколения для делового использования на самых разнообразных компьютерах — от переносных компьютеров до высококлассных серверов. Данная операционная система основывается на технологии NT и является наилучшей операционной системой для ведения коммерческой деятельности в Интернете. Система является надежной: настольные компьютеры, портативные компьютеры и серверы, на которых используется операционная система Windows 2000, работают безотказно. Применение Windows 2000 снижает затраты, так как упрощается управление системой. Кроме того, это наилучшая операционная система, которая позволяет применять любое новейшее оборудование —

от самых маленьких мобильных устройств и до самых больших серверов для электронной коммерции.

Операционная система Windows 2000 Professional объединяет присущую Windows 98 простоту использования в Интернете, на работе, в пути, с присущими Windows NT

управляемостью, надежностью и безопасностью.

Семейство Windows 2000 Server является новым поколением удостоенной наград и пользующейся коммерческим успехом операционной системы Windows NT Server. Данное семейство состоит из многоцелевых масштабируемых сетевых операционных систем. Windows 2000 предоставит удобную в управлении систему, обеспечивающую большую работоспособность оборудования, а также платформу для наиболее ответственных приложений электронной коммерции и ведения бизнеса в определенной области.

Windows 2000 Datacenter Server является самой производительной и полнофункциональной серверной операционной системой из всех, когда-либо предлагавшихся корпорацией Microsoft. Эта система поддерживает до 64 ГБ физической памяти, а также симметричную мультипроцессорную обработку с использованием до 32 процессоров. Ее стандартные компоненты обеспечивают 4-узловую кластеризацию и балансировку нагрузки. Она оптимизирована для работы с большими хранилищами данных, эконометрического анализа, моделирования крупномасштабных процессов в науке и технике, оперативной обработки транзакций и объединения серверов

Служба каталогов Microsoft Windows 2000 Active Directory является одним из самых важных новшеств операционной системы Windows 2000. Служба Active Directory значительно упрощает управление системой, усиливает систему безопасности и расширяет возможности интеграции с другими платформами. Ключевой особенностью, обеспечивающей интеграцию, является возможность синхронизации информации, хранимой в каталоге Active Directory, с другими хранилищами информации, в том числе со службой каталогов Novell Directory Services (NDS) и системными базами данных.

Русскоязычные версии Windows 2000 помимо полной поддержки русского языка имеют поддержку украинского, белорусского, казахского, армянского, грузинского, азербайджанского, узбекского, а также татарского языков. Это значит, что пользователи локализованной версии смогут создавать и редактировать документы на перечисленных языках, при том что интерфейс пользователя и справочная система будут оставаться на русском языке. Кроме того, в Windows 2000 локализованы и утилиты командной строки.

В Windows 2000 разработчики не только постарались учесть опыт создания NT-систем предыдущего поколения, сохранив все их традиционные достоинства, но и включили в нее много полезных наработок из привычной в своей доступности и простоте Windows 9x, как бы сблизив эти две разные системы.

Главное же и важнейшее ее достоинство для нас с вами – это совместимость с большинством программ Windows 9x. При этом надежность Windows 2000 на порядки выше, чем у Windows 9x.

Устойчивость работы Windows 2000 объясняется не только тем, что DOS в ней отсутствует – система полностью 32-х разрядная, но и тем, что в ней, в отличие от Windows 9x, применена так называемая вытесняющая многозадачность.

При таком способе реализации многозадачности, ни один процесс не сможет полностью завладеть центральным процессором, а получит в свое распоряжение лишь небольшой кусочек времени работы CPU, после чего процессор благополучно перейдет к обслуживанию следующего процесса – и так по кругу. Таким образом, каждый процесс обрабатывается по очереди под управлением специального диспетчера, и зависшая программа принудительно освобождает процессор, когда время, ей отведенное на работу, истекает. При появлении же сбоя достаточно снять “повисшую” задачу, что никак не отражается на деятельности всей системы и других программ, так как друг на друга они никак не влияют.

По сравнению с Windows NT, новая операционная система не только значительно облагорожена приятным внешним видом пользовательского интерфейса, который не вызовет никаких проблем у тех, кто видел Windows 9x, но и заметно улучшена поддержка широкого спектра нового оборудования. Система воспринимает без проблем и Plug and Play, USB, IEEE, ACPI, AGP, MMX, и даже FAT32.

Таким образом, наконец-то появилась операционная система, которая хоть как-то может

заменить нам “капризного” монополиста Windows 9x, позволив при этом не расстаться с любимыми программами. В большинстве случаев программы под Windows 2000 работают даже быстрее, чем под Windows 9x.

Главный недостаток Windows 2000 – большая требовательность к аппаратной конфигурации персонального компьютера, значительно превышающая запросы Windows 9x. И хотя Microsoft и заявляет, что минимум для нее – Pentium 133, 32Mb RAM, 2Gb HDD, на деле же – это характеристики машины, на которую Windows 2000 можно установить, но не работать.

В жизни же, рекомендуется как минимум 96Mb ОЗУ, при которых уже можно более-менее комфортно работать, лучше 128Mb и выше.

Процессор необходим не хуже Pentium 233МГц. 650 свободных мегабайт на жестком диске, как написано в руководстве Microsoft, также едва-едва хватит под саму операционную систему – под раздел с ОС надо отвести минимум 2-4Gb, иначе системные программы придется ставить в другие разделы.

Операционная система Linux

Linux – это операционная система для IBM-совместимых персональных компьютеров и рабочих станций. Это многопользовательская ОС с сетевой оконной графической системой X Window System. ОС Linux поддерживает стандарты открытых систем и протоколы сети Интернет и совместима с системами Unix, DOS, MS Windows. Все компоненты системы, включая исходные тексты, распространяются с лицензией на свободное копирование и установку для неограниченного числа пользователей.

Разработал эту операционную систему в начале 90-х годов тогда еще студент университета Хельсинки (Финляндия), Линус Торвалд при участии пользователей сети Интернет, сотрудников исследовательских центров, различных фондов и университетов (в том числе и МГИУ).

Будучи традиционной операционной системой, Linux (произносится «линукс», с ударением на первом слоге) выполняет многие из функций, характерных для DOS и Windows. Однако следует отметить, что эта ОС отличается особой мощностью и гибкостью. Система Linux разрабатывалась как ПК-версия операционной системы Unix, которая десятилетиями используется на мэйнфреймах и мини-ЭВМ и является основной ОС для рабочих станций. Linux предоставляет в распоряжение пользователя ПК скорость, эффективность и гибкость Unix, используя при этом все преимущества персональных машин. При работе с мышью активно используются все три кнопки, в частности средняя кнопка используется для вставки фрагментов текста.

С экономической точки зрения Linux обладает еще одним весьма существенным достоинством – это бесплатная система. Linux распространяется по генеральной открытой лицензии GNU в рамках фонда свободного программного обеспечения (Free Software Foundation), что делает эту ОС доступной для всех желающих. Linux защищена авторским правом и не находится в общедоступном пользовании, однако открытая лицензия GNU это почти то же самое, что и передача в общедоступное пользование. Она составлена так, что Linux остается бесплатной и в то же время стандартизированной системой. Существует лишь один официальный вариант ядра Linux.

От Unix операционной системе Linux достались еще две замечательные особенности: она является *многопользовательской* и *многозадачной* системой. Многозадачность означает, что система может выполнять несколько задач одновременно. Многопользовательский режим означает, что в системе могут одновременно работать несколько пользователей, каждый из которых взаимодействует с ней через свой терминал. Еще одним из достоинств этой ОС является возможность ее установки совместно с Windows на один компьютер.

Linux способен любую персональную машину превратить в рабочую станцию. В наше время Linux является операционной системой для бизнеса, образования и индивидуального программирования. Университеты по всему миру применяют Linux в учебных курсах по программированию и проектированию операционных систем. Он стал незаменим в широких корпоративных сетях, а также для организации Интернет-узлов и Web-серверов.

Современный Linux предоставляет возможность использовать несколько разновидностей графического интерфейса: *KDE* (K Desktop Environment), *GNOME* (GNU Network Model Environment) и другие.

В каждой из этих оболочек пользователю предоставляется возможность работы сразу с несколькими рабочими столами (в то время как в MS Windows всегда один рабочий стол, который

приходится загромождать окнами).

Способы построения современных операционных систем и операционных оболочек

Рассмотрим какие существуют подходы к построению операционных систем:

Монолитное ядро

По сути дела, операционная система – это обычная программа, поэтому было бы логично и организовать ее так же, как устроено большинство программ, то есть составить из процедур и функций. В этом случае компоненты операционной системы являются не самостоятельными модулями, а составными частями одной большой программы. Такая структура операционной системы называется монолитным ядром (monolithic kernel). Монолитное ядро представляет собой набор процедур, каждая из которых может вызвать каждую. Все процедуры работают в привилегированном режиме. Таким образом, монолитное ядро – это такая схема операционной системы, при которой все ее компоненты являются составными частями одной программы, используют общие структуры данных и взаимодействуют друг с другом путем непосредственного вызова процедур. Для монолитной операционной системы ядро совпадает со всей системой.

Во многих операционных системах с монолитным ядром сборка ядра, то есть его компиляция, осуществляется отдельно для каждого компьютера, на который устанавливается операционная система. При этом можно выбрать список оборудования и программных протоколов, поддержка которых будет включена в ядро. Так как ядро является единой программой, перекомпиляция – это единственный способ добавить в него новые компоненты или исключить неиспользуемые. Следует отметить, что присутствие в ядре лишних компонентов крайне нежелательно, так как ядро всегда полностью располагается в оперативной памяти. Кроме того, исключение ненужных компонентов повышает надежность операционной системы в целом.

Монолитное ядро – старейший способ организации операционных систем. Примером систем с монолитным ядром является большинство Unix-систем.

Даже в монолитных системах можно выделить некоторую структуру. Как в бетонной глыбе можно различить вкрапления щебенки, так и в монолитном ядре выделяются вкрапления сервисных процедур, соответствующих системным вызовам. Сервисные процедуры выполняются в привилегированном режиме, тогда как пользовательские программы – в непривилегированном. Для перехода с одного уровня привилегий на другой иногда может использоваться главная сервисная программа, определяющая, какой именно системный вызов был сделан, корректность входных данных для этого вызова и передающая управление соответствующей сервисной процедуре с переходом в привилегированный режим работы. Иногда выделяют также набор программных утилит, которые помогают выполнять сервисные процедуры.

Многоуровневые системы (Layered systems)

Продолжая структуризацию, можно разбить всю вычислительную систему на ряд более мелких уровней с хорошо определенными связями между ними, так чтобы объекты уровня N могли вызывать только объекты уровня N-1. Нижним уровнем в таких системах обычно является hardware, верхним уровнем – интерфейс пользователя. Чем ниже уровень, тем более привилегированные команды и действия может выполнять модуль, находящийся на этом уровне. Впервые такой подход был применен при создании системы THE (Technische Hogeschool Eindhoven) Дейкстрой (Dijkstra) и его студентами в 1968 г. Эта система имела следующие уровни:

5	Интерфейс пользователя
4	Управление вводом выводом
3	Драйвер устройства связи оператора и консоли
2	Управление печатью
1	Планирование задач и процессов
0	Hardware

Рис. 1.2. Слоеная система THE

Слоеные системы хорошо реализуются. При использовании операций нижнего слоя не нужно знать, как они реализованы, нужно лишь понимать, что они делают. Слоеные системы хорошо

тестируются. Отладка начинается с нижнего слоя и проводится послойно. При возникновении ошибки мы можем быть уверены, что она находится в тестируемом слое. Слоеные системы хорошо модифицируются. При необходимости можно заменить лишь один слой, не трогая остальные. Но слоеные системы сложны для разработки: тяжело правильно определить порядок слоев и что к какому слою относится. Слоеные системы менее эффективны, чем монолитные. Так, например, для выполнения операций ввода-вывода программе пользователя придется последовательно проходить все слои от верхнего до нижнего.

Виртуальные машины

Пусть операционная система реализует виртуальную машину для каждого пользователя, но не упрощая ему жизнь, а, наоборот, усложняя. Каждая такая виртуальная машина предстает перед пользователем как голое железо – копия всего hardware в вычислительной системе, включая процессор, привилегированные и непривилегированные команды, устройства ввода-вывода, прерывания и т.д. И он остается с этим железом один на один. При попытке обратиться к такому виртуальному железу на уровне привилегированных команд в действительности происходит системный вызов реальной операционной системы, которая и производит все необходимые действия. Такой подход позволяет каждому пользователю загрузить свою операционную систему на виртуальную машину и делать с ней все, что душа пожелает.

Программа пользователя	Программа пользователя	Программа пользователя
MS-DOS	Linux	Windows-NT
Виртуальное hardware	Виртуальное hardware	Виртуальное hardware
Реальная операционная система		
Реальное hardware		

Рис. 1.3. Вариант виртуальной машины

Первой реальной системой такого рода была система CP/CMS, или VM/370, как ее называют сейчас, для семейства машин IBM/370.

Недостатком таких операционных систем является снижение эффективности виртуальных машин по сравнению с реальным компьютером, и, как правило, они очень громоздки. Преимущество же заключается в использовании на одной вычислительной системе программ, написанных для разных операционных систем.

Микроядерная архитектура

Современная тенденция в разработке операционных систем состоит в перенесении значительной части системного кода на уровень пользователя и одновременной минимизации ядра. Речь идет о подходе к построению ядра, называемом микроядерной архитектурой (microkernel architecture) операционной системы, когда большинство ее составляющих являются самостоятельными программами. В этом случае взаимодействие между ними обеспечивает специальный модуль ядра, называемый микроядром. Микро-ядро работает в привилегированном режиме и обеспечивает взаимодействие между программами, планирование использования процессора, первичную обработку прерываний, операции ввода-вывода и базовое управление памятью.

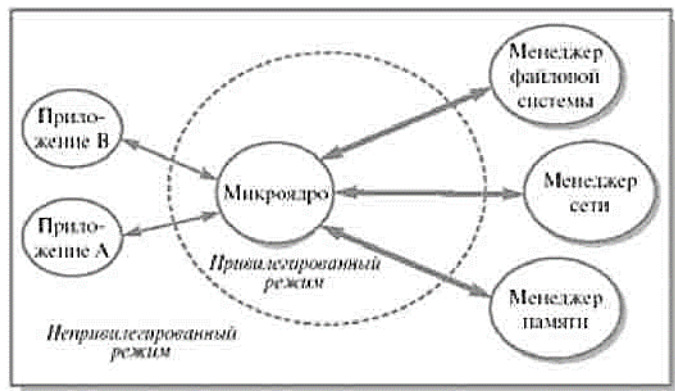


Рис. 1.4. Микроядерная архитектура операционной системы

Остальные компоненты системы взаимодействуют друг с другом путем передачи сообщений через микроядро.

Основное достоинство микроядерной архитектуры – высокая степень модульности ядра операционной системы. Это существенно упрощает добавление в него новых компонентов. В микроядерной операционной системе можно, не прерывая ее работы, загружать и выгружать новые драйверы, файловые системы и т. д. Существенно упрощается процесс отладки компонентов ядра, так как новая версия драйвера может загружаться без перезапуска всей операционной системы. Компоненты ядра операционной системы ничем принципиально не отличаются от пользовательских программ, поэтому для их отладки можно применять обычные средства. Микроядерная архитектура повышает надежность системы, поскольку ошибка на уровне непривилегированной программы менее опасна, чем отказ на уровне режима ядра.

В то же время микроядерная архитектура операционной системы вносит дополнительные накладные расходы, связанные с передачей сообщений, что существенно влияет на производительность. Для того чтобы микроядерная операционная система по скорости не уступала операционным системам на базе монолитного ядра, требуется очень аккуратно проектировать разбиение системы на компоненты, стараясь минимизировать взаимодействие между ними. Таким образом, основная сложность при создании микроядерных операционных систем – необходимость очень аккуратного проектирования.

Смешанные системы

Все рассмотренные подходы к построению операционных систем имеют свои достоинства и недостатки. В большинстве случаев современные операционные системы используют различные комбинации этих подходов. Так, например, ядро операционной системы Linux представляет собой монолитную систему с элементами микроядерной архитектуры. При компиляции ядра можно разрешить динамическую загрузку и выгрузку очень многих компонентов ядра – так называемых модулей. В момент загрузки модуля его код загружается на уровне системы и связывается с остальной частью ядра. Внутри модуля могут использоваться любые экспортируемые ядром функции.

Другим примером смешанного подхода может служить возможность запуска операционной системы с монолитным ядром под управлением микроядра. Так устроены 4.4BSD и MkLinux, основанные на микроядре Mach. Микроядро обеспечивает управление виртуальной памятью и работу низкоуровневых драйверов. Все остальные функции, в том числе взаимодействие с прикладными программами, осуществляется монолитным ядром. Данный подход сформировался в результате попыток использовать преимущества микроядерной архитектуры, сохраняя по возможности хорошо отлаженный код монолитного ядра.

Наиболее тесно элементы микроядерной архитектуры и элементы монолитного ядра переплетены в ядре Windows NT. Хотя Windows NT часто называют микроядерной операционной системой, это не совсем так. Микроядро NT слишком велико (более 1 Мбайт), чтобы носить приставку "микро". Компоненты ядра Windows NT располагаются в вытесняемой памяти и взаимодействуют друг с другом путем передачи сообщений, как и положено в микроядерных операционных системах. В то же время все компоненты ядра работают в одном адресном пространстве и активно используют общие структуры данных, что свойственно операционным системам с монолитным ядром. По мнению специалистов Microsoft, причина проста: чисто микроядерный дизайн коммерчески невыгоден, поскольку неэффективен.

Таким образом, Windows NT можно с полным правом назвать гибридной операционной системой. Многопроцессорные ОС разделяют на симметричные и асимметричные. В симметричных ОС на каждом процессоре функционирует одно и то же ядро, и задача может быть выполнена на любом процессоре, то есть обработка полностью децентрализована. При этом каждому из процессоров доступна вся память.

В асимметричных ОС процессоры неравноправны. Обычно существует главный процессор (master) и подчиненные (slave), загрузку и характер работы которых определяет главный процессор.

Системы реального времени

В разряд многозадачных ОС, наряду с пакетными системами и системами разделения времени, включаются также системы реального времени.

Они используются для управления различными техническими объектами или технологическими процессами. Такие системы характеризуются предельно допустимым временем реакции на внешнее событие, в течение которого должна быть выполнена программа, управляющая

объектом. Система должна обрабатывать поступающие данные быстрее, чем они могут поступать, причем от нескольких источников одновременно.

Столь жесткие ограничения сказываются на архитектуре систем реального времени, например, в них может отсутствовать виртуальная память, поддержка которой дает непредсказуемые задержки в выполнении программ.

Приведенная классификация ОС не является исчерпывающей.

Организация и управление памятью, распределение ресурсов, сервисные службы операционных систем, организация сохранности и защиты программных систем

Основную (оперативную) *память* компьютерной системы можно рассматривать как большой *массив* слов или байтов, каждый из которых имеет свой *адрес*. *Память* - это *хранилище данных* с быстрым доступом, совместно используемое процессором и *устройствами ввода-вывода*.

Основная память – это неустойчивое (volatile) устройство памяти. Ее содержимое теряется при сбое системы или при выключении питания. Для организации устойчивой памяти используются другие, более медленные технологии.

ОС отвечает за следующие действия, связанные с управлением памятью:

- Отслеживание того, какие части памяти в данный момент используются и какими процессами. Как правило, ОС организует для каждого процесса свою виртуальную память – расширение основной памяти путем хранения ее образа на диске и организации подкачки в *основную память* фрагментов (страниц или сегментов) виртуальной памяти процесса и ее откачки по мере необходимости.

- Стратегия загрузки процессов в основную память, по мере ее освобождения. При активизации процесса и его запуске или продолжении его выполнения процесс должен быть загружен в *основную память*, что и осуществляется операционной системой. При этом, возможно, какие-либо не активные в данный момент процессы приходится откачивать на диск.

- Выделение и освобождение памяти по мере необходимости. ОС обслуживает запросы вида "выделить область основной памяти длиной n байтов" и "освободить область памяти, начинающуюся с заданного адреса, длиной m байтов". Длина участков выделяемой и освобождаемой памяти может быть различной. ОС хранит список занятой и свободной памяти. При интенсивном использовании памяти может возникнуть ее фрагментация – дробление на мелкие свободные части, вследствие того, что при запросах на выделение памяти длина найденного сегмента оказывается немного больше, чем требуется, и остаток сохраняется в списке свободной памяти как область небольшого размера (подчас всего 1 – 2 слова). В курсе рассмотрены различные стратегии управления памятью и борьбы с фрагментацией. При исчерпании основной памяти ОС выполняет сборку мусора – поиск не используемых фрагментов, на которые потеряны ссылки, и уплотнение (компактировку) памяти – сдвиг всех используемых фрагментов по меньшим адресам, с корректировкой всех адресов.

Поскольку размер основной памяти недостаточен для постоянного хранения всех программ и данных, в компьютерной системе должна быть предусмотрена **вторичная (внешняя) память** для откачки (*back up, swapping*) части содержимого основной памяти.

В большинстве компьютерных систем в качестве главной вторичной памяти для хранения программ и данных используются диски.

ОС отвечает за выполнение следующих действий, связанных с управлением дисками:

- Управление свободной дисковой памятью;
- Выделение дисковой памяти;
- Диспетчеризация дисков (disk scheduling).

При управлении вторичной памятью возникают проблемы, аналогичные проблемам распределения основной памяти. Всякая *память*, даже самая большая *по* объему, рано или поздно может исчерпаться, либо фрагментироваться на множество мелких областей свободной памяти. О *методах управления* основной и внешней памятью речь пойдет подробно ниже в специальных разделах курса.

Управление сетевыми (распределенными) системами. Как уже было сказано в более ранних лекциях, распределенная система – это совокупность процессоров, которые не используют *общую*

память или часы (такты процессора). Каждый *процессор* имеет собственную *локальную память*. Процессоры в такой системе соединены в *сеть*. Сетевое взаимодействие выполняется по определенному протоколу (интерфейсу, набору операций). Наиболее распространенный *сетевой протокол* – TCP/IP, основанный на IP-адресах машин (hosts); например, 190.100.125.1.

В распределенной системе ОС обеспечивает *доступ* пользователей к различным общим сетевым ресурсам – например, файловым системам или принтерам. Каждому общему ресурсу ОС присваивает определенное сетевое имя и управляет возможностью доступа к нему с различных компьютеров сети. ОС обеспечивает также удаленный запуск программ на другом компьютере сети – возможность входа на другой *компьютер* и работы на нем, с использованием памяти, процессора и диска удаленной, как правило, более мощной машины, и использованием клиентского компьютера в качестве терминала. В *Windows* такая возможность называется удаленный рабочий стол (remote desktop connection).

Доступ к общим ресурсам (*shared resource*) в распределенной системе позволяет:

- Ускорить вычисления;
- Расширить границы доступа к данным;
- Обеспечить более высокую надежность.

Система защиты (protection)

Термин защита (protection) используется для обозначения механизма управления доступом программ, процессов и пользователей к системным и *пользовательским ресурсам*.

Механизм защиты в ОС должен обеспечивать следующие возможности:

Различать авторизованный, или санкционированный (authorized) и несанкционированный (unauthorized) *доступ*. Под авторизацией понимается предоставление операционной системой пользователю или программе какого-либо определенного набора полномочий (permissions), например, возможности чтения или изменения файлов в файловой *системе с общим доступом*.

Описывать предназначенные для защиты элементы *управления (конфигурации)*. Например, в *UNIX* используются специальные текстовые конфигурационные файлы для представления информации о файловых системах, к которым возможен сетевой *доступ*, с указанием списка машин (хостов), с которых возможен *доступ*, и набора действий, которые могут быть выполнены.

Обеспечивать средства выполнения необходимых для защиты действий (сигналы, исключения, *блокировка* и др.). Например, система защиты ОС должна фильтровать *сетевые пакеты*, получаемые извне локальной сети, выбирать и отсеивать "неблагонадежные" (получаемые с подозрительных IP-адресов), сообщать пользователю об обнаруженных и ликвидированных попытках *сетевых атак* с целью "взлома" Вашего компьютера (что и происходит на практике, например, при работе в *Windows*, когда Вы выходите в *Интернет* с Вашего компьютера). Если Вы нарушили условия защиты (например, Ваша *программа* попыталась обратиться к файлу, работать с которым у Вас нет полномочий), ОС должна выдать понятное сообщение и прекратить работу программы. В современных системах это делается с помощью генерации исключений (exceptions), например, SecurityException.

Система поддержки командного интерпретатора

Большинство команд для ОС задаются с помощью специальных управляющих операторов, предназначенных для выполнения следующих основных функций:

- создания процессов и управления процессами; например, в UNIX команда *ps* –а выводит в *стандартный вывод* процесса информацию обо всех *активных процессах* в системе, с указанием их номеров (*PID*);
- выполнения ввода-вывода; например, в системе MS DOS команда *type file_name* выполняет вывод на терминал содержимого заданного текстового файла;
- управления вторичной памятью; например, в UNIX команда *share /mydir* добавляет директорию */mydir* к списку совместно используемых в локальной сети файловых систем;
- управления основной памятью; например, команда *swar* в ОС Solaris позволяет управлять размером пространства дисковой памяти для реализации виртуальной памяти (*swap*) и выводить информацию о его состоянии;
- доступа к файловой системе; например, в большинстве ОС команда *cd new_dir* устанавливает заданную директорию в качестве текущей (рабочей);

- защиты; например, в системе UNIX команда `chmod 700 my_home_dir` защитит Вашу домашнюю директорию от непрошенных любопытных глаз – "лазутчик" не сможет даже выполнить команду `cd` для этой директории и, тем более, читать в ней какие-либо файлы;
- управления сетью; например, команды `telnet host_name` и `rlogin host_name` (последняя доступна в системе UNIX) служат для удаленного входа на другой компьютер сети.

Программа, которая читает и интерпретирует *операторы управления*, называется командным интерпретатором. В *Windows* это *интерпретатор command.com*, доступный для выполнения команд в окне MS DOS prompt. В *UNIX*, *Linux*, *Solaris* это уже упоминавшиеся всевозможные "шеллы": `sh`, `csh`, `ksh`, `bash` – процессоры для интерпретации мощных командных языков. Функция командного процессора состоит в том, чтобы прочесть и исполнить очередной управляющий оператор (команду).

Сервисы (службы) ОС

Операционная система предоставляет для пользователей *целый ряд сервисных возможностей*, или, коротко, сервисов (служб):

Исполнение программ – *загрузка* программы в *память* и ее выполнение; например, в *Windows* при запуске программы ОС находит в файле ее двоичного кода (`.exe`) так называемую загрузку для исполнения (*execution stub*), содержащую ссылку на код головного метода `main`, и запускает его. В среде *.NET* этот же *execution stub* в файле двоичного кода используется системой для вызова не непосредственно исполняемой программы, а общего окружения времени выполнения – *Common Language Runtime (CLR)*, которое обеспечивает особый режим (*managed execution*) выполнения программы.

Поддержка ввода-вывода – обеспечение интерфейса для работы программ с *устройствами ввода-вывода*. Например, в *UNIX* у каждой программы есть свой *стандартный ввод* и *стандартный вывод* (по умолчанию это *терминал*). В более старых ОС, например, *IBM 360*, привязку программы к устройствам ввода-вывода требовалось специфицировать с помощью громоздких *DD (Data Definition)* – предложений на специальном языке *управления заданиями*.

Работа с файловой системой – предоставление программам интерфейса для создания, именования, удаления *файлов*.

Коммуникация – *обмен информацией* между процессами, выполняемыми на одном компьютере или на других системах, связанных в *сеть*. В операционных системах реализуется с помощью общей памяти (*shared memory*) или передачи сообщений.

Обнаружение ошибок в работе *процессора*, *памяти*, *устройств ввода-вывода* и программах пользователей.

Дополнительные функции ОС

Данные функции реализованы не непосредственно для удобства пользователя, а для обеспечения выполнения операций системы. Это следующие возможности.

Распределение ресурсов между пользователями, программами и процессами, работающими одновременно.

Ведение статистики использования ресурсов, с целью выставления пользователям счетов (например, за сетевой трафик) или для анализа эффективности работы системы.

Защита – обеспечение того, чтобы *доступ* к любым ресурсам был контролируемым.

Языки и системы программирования. Модели языков программирования

Языки и системы программирования. Модели языков программирования.

Системы программирования - это комплекс инструментальных программных средств, предназначенный для работы с программами на одном из языков программирования. Системы программирования предоставляют сервисные возможности программистам для разработки их собственных компьютерных программ.

В настоящее время разработка любого системного и прикладного программного обеспечения осуществляется с помощью систем программирования, в состав которых входят

- трансляторы с языков высокого уровня;
- средства редактирования, компоновки и загрузки программ;
- макроассемблеры (машинно-ориентированные языки);
- отладчики машинных программ.

Системы программирования, как правило, включают в себя

- текстовый редактор (**Edit**), осуществляющий функции записи и редактирования исходного текста программы;
- загрузчик программ (**Load**), позволяющий выбрать из директория нужный текстовый файл программы;
- запускатель программ (**Run**), осуществляющий процесс выполнения программы;
- компилятор (**Compile**), предназначенный для компиляции или интерпретации исходного текста программы в машинный код с диагностикой синтаксических и семантических (логических) ошибок;
- отладчик (**Debug**), выполняющий сервисные функции по отладке и тестированию программы;
- диспетчер файлов (**File**), предоставляющий возможность выполнять операции с файлами: сохранение, поиск, уничтожение и т.п.

Язык программирования - Формализованный язык, предназначенный для описания программ и алгоритмов решения задач на ЭВМ. Языки программирования являются искусственными. В них синтаксис и семантика строго определены. Поэтому они не допускают свободного толкования выражения, что характерно для естественного языка.

Существующие языки программирования можно разделить на две группы: процедурные и непроцедурные, рис. 2.9.

Декларативный (непроцедурный) язык - позволяет задавать связи и отношения между объектами и величинами, но не определяет последовательность выполнения действий (например, языки Пролог, QBE);

Процедурные (алгоритмические) программы представляют из себя систему предписаний для решения конкретной задачи. Роль компьютера сводится к механическому выполнению этих предписаний.

Процедурные языки разделяют на языки *низкого и высокого уровня*.

Языки низкого уровня (машинно-ориентированные) позволяют создавать программы из машинных кодов, обычно в шестнадцатиричной форме. С ними трудно работать, но созданные с их помощью высококвалифицированным программистом программы занимают меньше места в памяти и работают быстрее. С помощью этих языков удобнее разрабатывать системные программы, драйверы (программы для управления устройствами компьютера), некоторые другие виды программ.

Язык высокого уровня средства которого обеспечивают описание задачи в наглядном, легко воспринимаемом виде, удобном для программиста. Он не зависит от внутренних машинных кодов ЭВМ любого типа, поэтому программы, написанные на языках высокого уровня, требуют перевода в **машинные коды** программами транслятора либо интерпретатора. К языкам высокого уровня относят Фортран, ПЛ/1, Бейсик, Паскаль, Си, Ада и др.



Рис. 2.9. Общая классификация языков программирования

Языки высокого уровня не зависят от архитектуры компьютера. Чем более язык ориентирован на человека, тем выше его уровень.

Модели языков программирования

Модульная модель языков программирования

Рис. 21.1

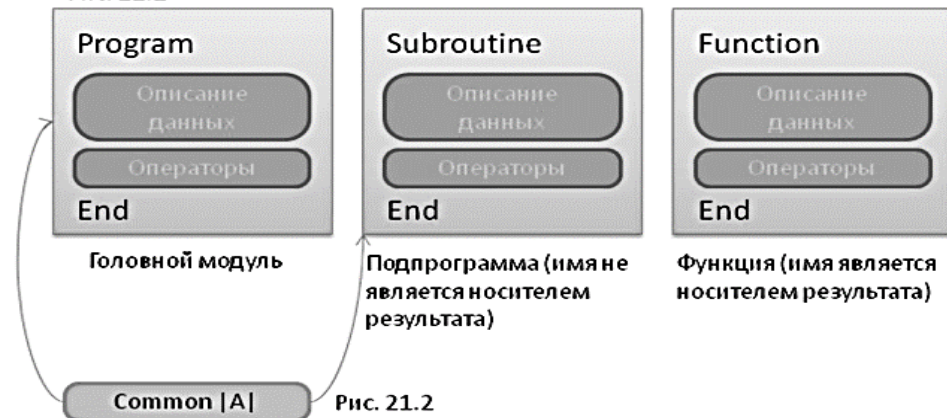


Рис. 21.2

Модульная модель:

- Каждая подпрограмма (или функция) является самостоятельным модулем.
- Определенные внутри модуля переменные локализованы в нём (не глобализованы)
- Память на объекты (переменные) выделяется статически
- Наличие областей памяти, разделяемых различными подпрограммами (common-блок) – имитация глобальной среды (Рис. 21.2)

Принцип модульности ляжет практически во все языки программирования. Нам хочется, чтобы модуль проявлялся как надклассовый объект. Сейчас понятие функции и модуля тождественно.

Блочная модель языков программирования

Центр разработки находился в Дубне. Сильно использовался в СССР. Теоретическая база – Алгол, но более практическая – Фортран. Си, Паскаль – «обглоданный» Алгол.

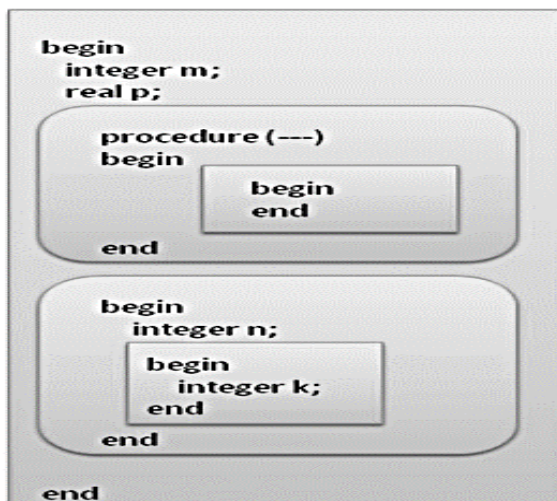


Рис 21.3

Дикая по сложности иерархия. Можно определять функцию в функции. Блоки имеют неограниченную глубину вложения. Вся программа представляет собой единый моноблок.

- Существует единственный программный «модуль»
- Вся программа представляет собой систему вложенных блоков
- Процедуры (как и ранее обобщение функции и подпрограммы) и функции вкладываются в блоки, их тела являются полноценными блоками
- Данные локализованы в блоках
- Внутренний блок наследует данные объемлющих его блоков
- Память под переменные выделяется при входе в блок и удаляется при выходе из него (т.е. полная динамика – статическая модель отсутствует)

На Алголе - ГДР написаны практически все ядерные исследования и исследования в квантовой химии. На Си, и более современных постАлгольных языках такие задачи реализованы не были.

Серьёзных методов агрегирования на Алголе и Фортране не были. Массивы были и только они, более сложных структур не было.

Блочная - модульная архитектура (Си)

Сюда приходит блочная структура, но сильно лимитированная. Блоки есть, но нельзя создавать операцию одну в другой.

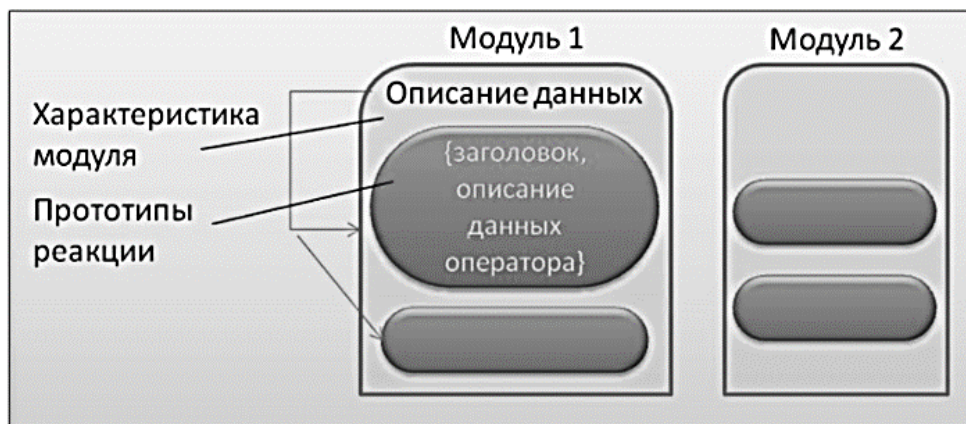


Рис.22.1.

Все активные компоненты вкладываются внутрь модуля.

Модулей уже не один.

Функцию в функции мы создать уже не можем, но блоки у нас уже есть.

Компиляторы и интерпретаторы

Основным модулем системы программирования всегда является компилятор. Именно технические характеристики компилятора, прежде всего, влияют на эффективность результирующих программ, порождаемых системой программирования.

Транслятор – это программа, которая переводит входную программу на исходном (входном) языке в эквивалентную ей выходную программу на результирующем (выходном) языке.

Близко по смыслу к этому понятию понятие компилятор.

Компилятор – это транслятор, который осуществляет перевод исходной программы в эквивалентную ей объектную программу на языке машинных команд или языке ассемблера (.exe файл). Таким образом, компилятор отличается от транслятора тем, что его результирующая программа написана обязательно на языке машинных команд или языке ассемблера. Результирующая программа транслятора в общем случае может быть написана на любом языке (например, транслятор с языка Pascal на язык C). Таким образом, компиляторы – это вид трансляторов.

Существует еще принципиально отличное понятие: интерпретатор.

Интерпретатор – это программа, которая воспринимает входную программу на исходном языке, переводит каждый оператор или строку в машинный язык и выполняет их. Интерпретатор не порождает результирующую программу и никакого результирующего кода.

Ядро системы программирования составляет язык.

Объектно-ориентированное программирование

Принципиально иное направление в программировании связано с методологиями непроцедурного программирования. К ним можно отнести **объектно-ориентированное программирование**. Программы стали строиться не из процедур и функций, а из сравнительно простых кирпичиков-объектов, в которых были упрятаны данные и подпрограммы их обработки. Программирование рассматриваемого стиля заключается в выборе имеющихся или создании новых объектов и организации взаимодействия между ними.

Определения

Объект – это то, чем вы управляете с помощью программы (например в Excel - ячейка, диапазон, диаграмма и т.п. В БД- это таблицы, формы, запросы). Каждый объект имеет свой **тип (класс)**. Класс представляет собой **тип данных**, имеющий в составе данный объект.

Свойства - Параметры объекта (конечно, не все, а только необходимые в программе).

Методы - Действия, которые можно выполнять над объектом данного типа, или которые сам объект может выполнять. Это процедуры и функции.

Инкапсуляция. Инкапсуляция — это объединение данных и действий над ними. Это принцип, согласно которому любой **класс** должен рассматриваться как **чёрный ящик** — пользователь класса должен видеть и использовать только интерфейс (от английского interface — внешнее лицо, т. е. список декларируемых свойств и методов) **класса** и не вникать в его внутреннюю реализацию. Это свойство объектов, заключающееся в сокрытии информации о внутренней "жизни" объекта. Т.е. внешний пользователь объекта может использовать его возможности посредством определенного интерфейса, а непосредственно работа с данными и детали ее реализации от него скрыты. Так, в примере с жестким диском, нам для работы с ним достаточно знать, что он может хранить информацию, и позволяет записывать и считывать ее. При этом вовсе необязательно разбираться в том, каким образом хранится информация и как протекают процессы ее записи и чтения. Свойство инкапсуляции облегчает написание программ, и, что самое главное, их модификацию.

Наследование. Наследованием называется возможность порождать один **класс** от другого с сохранением всех свойств и методов класса-предка и добавляя, при необходимости, новые свойства и методы. Наследование призвано отобразить такое свойство реального мира, как иерархичность.

Полиморфизм Полиморфизмом называют явление, при котором для различных родственных объектов можно задать одинаковое название функции, но действия функция будет выполнять разные, в зависимости от конкретного объекта. Например, надо нарисовать прямоугольник или круг, в зависимости от типов: «прямоугольник», «круг». Но в обоих случаях функция будет иметь имя «Нарисовать». Т.е. действия с одинаковым именем вызывает различное поведение, в зависимости от типа данных.

К наиболее современным объектно-ориентированным языкам программирования относятся **C++ и Java**.

Языки C++ и Java. За основу языка C++ был взят язык C, дополненный элементами языков BCPL, Simula-67 и Algol-68.

Новая интегрируемая в Internet версия языка, получила название **Java**. С января 1995 года Java получает распространение в Internet. Принципиальной разницей между Java и C++ является то, что первый из них является интерпретируемым, а второй — компилируемым. Синтаксис языков практически полностью совпадает. С точки зрения возможностей собственно объектно-ориентированных средств язык Java обладает рядом преимуществ перед языком C++. Так, язык Java демонстрирует более гибкую и мощную систему, кроме того, он проще и надежнее.

В силу своей конструктивности идеи объектно-ориентированного программирования используются во многих универсальных процедурных языках, в которые входит специальная

библиотека объектно-ориентированного программирования. Это системы программирования **TurboVision, VisualBasic, Delphi** и др.

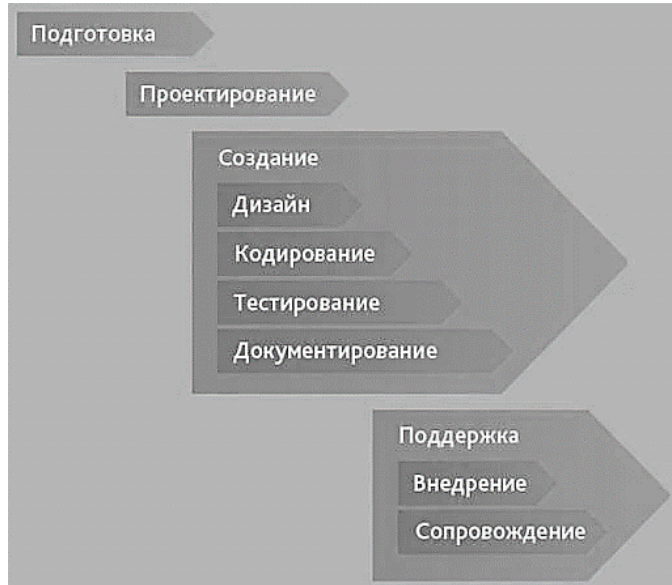
Visual Basic подходит для написания небольших и нетребовательных к ресурсам программ. А так как он является языком создания приложений Microsoft Office, то он получил самое широкое распространение.

Delphi является очередным шагом в эволюции компиляторов Паскаля. Среда **Delphi** стала, по сути, лучшим средством программирования для операционной системы Windows. У языка **Delphi** есть еще одно очень важное преимущество перед остальными коммерчески успешными языками — он великолепно подходит для обучения программированию. Поэтому его рекомендуют в качестве первого языка для всех учеников и студентов, собирающихся стать профессиональными программистами.

Delphi, Java применяются для создания программ, используемых в Интернете.

Модели и этапы разработки программного обеспечения

У любого программного обеспечения есть жизненный цикл — этапы, через которые оно проходит с начала создания до конца разработки и внедрения. Чаще всего это подготовка, проектирование, создание и поддержка. Этапы могут называться по-разному и дробиться на более мелкие стадии.



Рассмотрим эти **этапы** на примере жизненного цикла интернет-магазина.

Подготовка. Иван решил запустить книжный интернет-магазин и начал анализировать, какие подобные сайты уже представлены в сети. Собрал информацию об их трафике, функциональности.

Проектирование. Иван выбрал компанию-подрядчика и обсудил с её специалистами архитектуру и дизайн будущего интернет-магазина.

Создание. Иван заключил с разработчиками договор. Они начали писать код, отрисовывать дизайн, составлять документацию.

Поддержка. Иван подписал акт сдачи-приёмки, и подрядчик разместил интернет-магазин на «боевых» серверах. Пользователи начали его посещать и сообщать о замеченных ошибках в поддержку, а программисты — оперативно всё исправлять.

Модель разработки программного обеспечения описывает, какие стадии жизненного цикла оно проходит и что происходит на каждой из них.

А *методология* включает в себя набор методов по управлению разработкой: это правила, техники и принципы, которые делают её более эффективной.

Основные модели разработки ПО

- Code and fix — модель кодирования и устранения ошибок;
- Waterfall Model — каскадная модель, или «водопад»;
- V-model — V-образная модель, разработка через тестирование;
- Incremental Model — инкрементная модель;
- Iterative Model — итеративная (или итерационная) модель;
- Spiral Model — спиральная модель;
- Chaos model — модель хаоса;
- Prototype Model — прототипная модель.

Из этих моделей наиболее популярны пять основных: каскадная, V-образная, инкрементная, итерационная и спиральная. Разберём их подробнее.

Waterfall (каскадная модель, или «водопад»)

В этой модели разработка осуществляется поэтапно: каждая следующая стадия начинается только после того, как заканчивается предыдущая. Если всё делать правильно, «водопад» будет наиболее быстрой и простой моделью. Применяется уже почти полвека, с 1970-х.



Преимущества «водопада»

- *Разработку просто контролировать.* Заказчик всегда знает, чем сейчас заняты программисты, может управлять сроками и стоимостью.
- *Стоимость проекта определяется на начальном этапе.* Все шаги запланированы уже на этапе согласования договора, ПО пишется непрерывно «от и до».
- *Не нужно нанимать тестировщиков с серьёзной технической подготовкой.* Тестировщики смогут опираться на подробную техническую документацию.

Недостатки каскадной модели

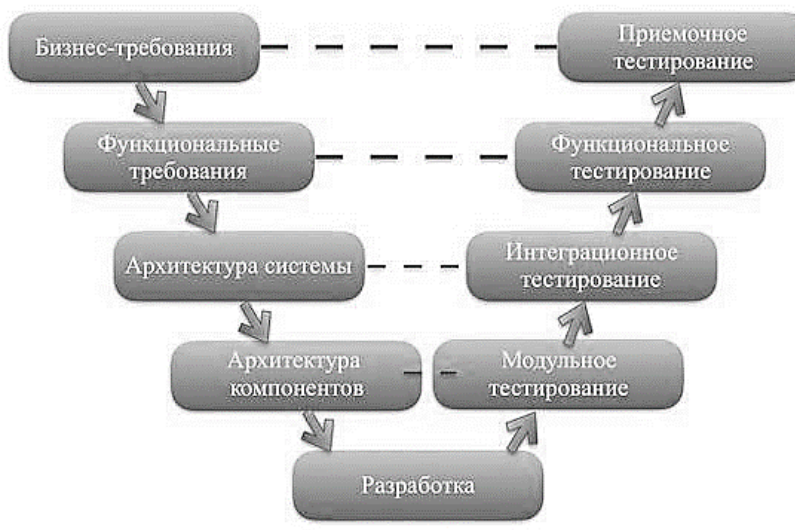
- *Тестирование начинается на последних этапах разработки.* Если в требованиях к продукту была допущена ошибка, то исправить её будет стоить дорого. Тестировщики обнаружат её, когда разработчик уже написал код, а технические писатели — документацию.
- *Заказчик видит готовый продукт в конце разработки и только тогда может дать обратную связь.* Велика вероятность, что результат его не устроит.
- *Разработчики пишут много технической документации, что задерживает работы.* Чем обширнее документация у проекта, тем больше изменений нужно вносить и дольше их согласовывать.

«Водопад» подходит для разработки проектов в *медицинской и космической отрасли, где уже сформирована обширная база документов (СНиПов и спецификаций)*, на основе которых можно написать требования к новому ПО.

При работе с каскадной моделью основная задача — написать подробные требования к разработке. На этапе тестирования не должно выясниться, что в них есть ошибка, которая влияет на весь продукт.

V-образная модель (разработка через тестирование)

Это усовершенствованная каскадная модель, в которой заказчик с командой программистов одновременно составляют требования к системе и описывают, как будут тестировать её на каждом этапе. История этой модели начинается в 1980-х.



Преимущества V-образной модели

- Количество ошибок в архитектуре ПО сводится к минимуму.

Недостатки V-образной модели

- Если при разработке архитектуры была допущена ошибка, то вернуться и исправить её будет стоить дорого, как и в «водопаде».

V-модель подходит для проектов, в которых важна надёжность и цена ошибки очень высока. Например, при разработке подушек безопасности для автомобилей или систем наблюдения за пациентами в клиниках.

Incremental Model (инкрементная модель)

Это модель разработки по частям (increment в переводе с англ. — приращение) уходит корнями в 1930-е. Рассмотрим её на примере создания социальной сети.

1. Заказчик решил, что хочет запустить соцсеть, и написал подробное техническое задание. Программисты предложили реализовать основные функции — страницу с личной информацией и чат. А затем протестировать на пользователях, «взлетит или нет».
2. Команда разработки показывает продукт заказчику и выпускает его на рынок. Если и заказчику, и пользователям социальная сеть нравится, работа над ней продолжается, но уже по частям.
3. Программисты параллельно создают функциональность для загрузки фотографий, обмена документами, прослушивания музыки и других действий, согласованных с заказчиком. Инкремент за инкрементом они совершенствуют продукт, приближаясь к описанному в техническом задании.



Преимущества инкрементной модели

- Не нужно вкладывать много денег на начальном этапе. Заказчик оплачивает создание основных функций, получает продукт, «выкатывает» его на рынок — и по итогам обратной связи решает, продолжать ли разработку.
- Можно быстро получить фидбэк от пользователей и оперативно обновить техническое задание. Так снижается риск создать продукт, который никому не нужен.

- *Ошибка обходится дешевле.* Если при разработке архитектуры была допущена ошибка, то исправить её будет стоить не так дорого, как в «водопаде» или V-образной модели.

Недостатки инкрементной модели

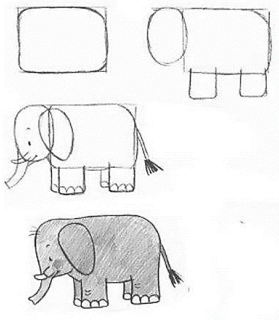
- *Каждая команда программистов разрабатывает свою функциональность и может реализовать интерфейс продукта по-своему.* Чтобы этого не произошло, важно на этапе обсуждения техзадания объяснить, каким он будет, чтобы у всех участников проекта сложилось единое понимание.

- *Разработчики будут оттягивать доработку основной функциональности и «пилить мелочёвку».* Чтобы этого не случилось, менеджер проекта должен контролировать, чем занимается каждая команда.

Инкрементная модель подходит для проектов, в которых точное техзадание прописано уже на старте, а продукт должен быстро выйти на рынок.

Iterative Model (итеративная модель)

Это модель, при которой заказчик не обязан понимать, какой продукт хочет получить в итоге, и может не прописывать сразу подробное техзадание.



Рассмотрим на примере создания мессенджера, как эта модель работает.

1. Заказчик решил, что хочет создать мессенджер. Разработчики сделали приложение, в котором можно добавить друга и запустить чат на двоих.
2. Мессенджер «выкатили» в магазин приложений, пользователи начали его скачивать и активно использовать. Заказчик понял, что продукт пользуется популярностью, и решил его доработать.
3. Программисты добавили в мессенджер возможность просмотра видео, загрузки фотографий, записи аудиосообщений. Они постепенно улучшают функциональность приложения, адаптируют его к требованиям рынка.

Преимущества итеративной модели

- *Быстрый выпуск минимального продукта* даёт возможность оперативно получать обратную связь от заказчика и пользователей. А значит, фокусироваться на наиболее важных функциях ПО и улучшать их в соответствии с требованиями рынка и пожеланиями клиента.

- *Постоянное тестирование пользователями* позволяет быстро обнаруживать и устранять ошибки.

Недостатки итеративной модели

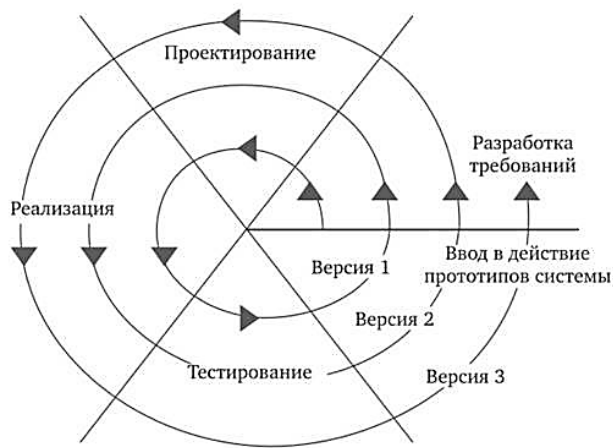
- *Использование на начальном этапе баз данных или серверов* — первые сложно масштабировать, а вторые не выдерживают нагрузку. Возможно, придётся переписывать большую часть приложения.

- *Отсутствие фиксированного бюджета и сроков.* Заказчик не знает, как выглядит конечная цель и когда закончится разработка.

Итеративная модель подходит для работы над большими проектами с неопределёнными требованиями, либо для задач с инновационным подходом, когда заказчик не уверен в результате.

Spiral Model (спиральная модель)

Используя эту модель, заказчик и команда разработчиков серьёзно анализируют риски проекта и выполняют его итерациями. Последующая стадия основывается на предыдущей, а в конце каждого витка — цикла итераций — принимается решение, продолжать ли проект. Эту модель начали использовать в 1988 году.



Рассмотрим, как функционирует эта модель, на примере разработки системы «Умный дом».

1. Заказчик решил, что хочет сделать такую систему, и заказал программистам реализовать управление чайником с телефона. Они начали действовать по модели «водопад»: выслушали идею, провели анализ предложений на рынке, обсудили с заказчиком архитектуру системы, решили, как будут её реализовывать, разработали, протестировали и «выкатили» конечный продукт.

2. Заказчик оценил результат и риски: насколько нужна пользователям следующая версия продукта — уже с управлением телевизором. Рассчитал сроки, бюджет и заказал разработку. Программисты действовали по каскадной модели и представили заказчику более сложный продукт, разработанный на базе первого.

3. Заказчик подумал, что пора создать функциональность для управления холодильником с телефона. Но, анализируя риски, понял, что в холодильник сложно встроить Wi-Fi-модуль, да и производители не заинтересованы в сотрудничестве по этому вопросу. Следовательно, риски превышают потенциальную выгоду. На основе полученных данных заказчик решил прекратить разработку и совершенствовать имеющуюся функциональность, чтобы со временем понять, как развивать систему «Умный дом».

Спиральная модель похожа на инкрементную, но здесь гораздо больше времени уделяется оценке рисков. С каждым новым витком спирали процесс усложняется. Эта модель часто используется в исследовательских проектах и там, где высоки риски.

Преимущества спиральной модели

- Большое внимание уделяется проработке рисков.

Недостатки спиральной модели

- Есть риск застрять на начальном этапе — бесконечно совершенствовать первую версию продукта и не продвинуться к следующим.
- Разработка длится долго и стоит дорого.

Программные средства и программные продукты

Программные средства – это набор программ, которые и заставляют аппаратную часть системы выполнять необходимые действия, «оживляют» компьютер. Эту часть компьютерной системы принято называть «software».

Программные средства (ПС) информационных технологий – это компьютерные (машинные) программы, представленные на языке программирования или в машинном коде описания действий, которые должна выполнить ЭВМ в соответствии с алгоритмом решения конкретной задачи или группы задач.

Программные средства информационных технологий на самом общем уровне делят на два класса: базовые ПС и прикладные ПС.

К базовым программным средствам, в свою очередь, относят:

- языки программирования;
- операционные системы (ОС);
- оболочки операционных систем;

- сервисные средства и утилиты.

Прикладные (специальные) программные средства (приложения) – это отдельные прикладные программы или пакеты прикладных программ, предназначенные для решения конкретных задач, связанных со сферой деятельности пользователей (управленческая, переводческая, проектно-конструкторская и т. п.), или конкретной предметной областью (проблемно-ориентированные информационные системы, БД).

Программный продукт – комплекс взаимосвязанных программ для решения определенной проблемы (задачи) массового спроса, подготовленный к реализации как любой вид промышленной продукции.

Путь от «программ для себя» до программных продуктов достаточно долгий, он связан с изменениями технической и программной среды разработки и эксплуатации программ, с появлением и развитием самостоятельной отрасли – информационного бизнеса, для которой характерны разделение труда фирм – разработчиков программ, их дальнейшая специализация, формирование рынка программных средств и информационных услуг.

Программные продукты могут создаваться как:

- индивидуальная разработка под заказ;
- разработка для массового распространения среди пользователей.

При индивидуальной разработке фирма-разработчик создает оригинальный программный продукт, учитывающий специфику обработки данных для конкретного заказчика.

При разработке для массового распространения фирма-разработчик, с одной стороны, должна обеспечить универсальность выполняемых функций обработки данных, с другой стороны, гибкость и настраиваемость программного продукта на условия конкретного применения. Отличительной особенностью программных продуктов должна быть их системность – функциональная полнота и законченность реализуемых функций обработки, которые применяются в совокупности.

Программный продукт разрабатывается на основе промышленной технологии выполнения проектных работ с применением современных инструментальных средств программирования. Специфика заключается в уникальности процесса разработки алгоритмов и программ, зависящего от характера обработки информации и используемых инструментальных средств. На создание программных продуктов затрачиваются значительные ресурсы – трудовые, материальные, финансовые; требуется высокая квалификация разработчиков.

Как правило, программные продукты требуют сопровождения, которое осуществляется специализированными фирмами – распространителями программ (дистрибьюторами), реже – фирмами-разработчиками. Сопровождение программ массового применения сопряжено с большими трудозатратами – исправление обнаруженных ошибок, создание новых версий программ и т.п.

Коммерческое, условно-бесплатное и свободно распространяемое программное обеспечение

Лицензионные программы. В соответствии с лицензионным соглашением разработчики программы гарантируют её нормальное функционирование в определенной операционной системе и несут за это ответственность.

Лицензионные программы разработчики обычно продают в коробочных дистрибутивах. В коробочке находятся CD-диски, с которых производится установка программы на компьютеры пользователей, и руководство пользователей по работе с программой.

Довольно часто разработчики предоставляют существенные скидки при покупке лицензий на использование программы на большом количестве компьютеров или учебных заведениях.

Условно бесплатные программы. Некоторые фирмы разработчики программного обеспечения предлагают пользователям условно бесплатные программы в целях рекламы и продвижения на рынок. Пользователю предоставляется версия программы с определённым сроком действия (после истечения указанного срока действия программы прекращает работать, если за неё не была произведена оплата) или версия программы с ограниченными функциональными

возможностями (в случае оплаты пользователю сообщается код, включающий все функции программы).

Производители бесплатного программного обеспечения заинтересованы в его широком распространении. К таким программным средствам можно отнести следующие:

Свободно распространяемые программы. Многие производители программного обеспечения и компьютерного оборудования заинтересованы в широком бесплатном распространении программного обеспечения. К таким программным средствам можно отнести:

- Новые недоработанные (бета) версии программных продуктов (это позволяет провести их широкое тестирование).
- Программные продукты, являющиеся частью принципиально новых технологий (это позволяет завоевать рынок).
- Дополнения к ранее выпущенным программам, исправляющие найденные ошибки или расширяющие возможности.
- Драйверы к новым или улучшенные драйверы к уже существующим устройствам.

Но какое бы программное обеспечение вы не выбрали, существуют общие требования ко всем группам программного обеспечения:

- Лицензионная чистота (применение программного обеспечения допустимо только в рамках лицензионного соглашения).
- Возможность консультации и других форм сопровождения.
- Соответствие характеристикам, комплектации, классу и типу компьютеров, а также архитектуре применяемой вычислительной техники.
- Надежность и работоспособность в любом из предусмотренных режимов работы, как минимум, в русскоязычной среде.
- Наличие интерфейса, поддерживающего работу с использованием русского языка. Для системного и инструментального программного обеспечения допустимо наличие интерфейса на английском языке.
- Наличие документации, необходимой для практического применения и освоения программного обеспечения, на русском языке.
- Возможность использования шрифтов, поддерживающих работу с кириллицей.

Наличие спецификации, оговаривающей все требования к аппаратным и программным средствам, необходимым для функционирования данного программного обеспечения.

Лицензионное программное обеспечение имеет ряд преимуществ:

Техническая поддержка производителя программного обеспечения. При эксплуатации приобретенного лицензионного программного обеспечения у пользователей могут возникнуть различные вопросы. Владельцы лицензионных программ имеют право воспользоваться технической поддержкой производителя программного обеспечения, что в большинстве случаев позволяет разрешить возникшие проблемы.

· Обновление программ. Производители программного обеспечения регулярно выпускают пакеты обновлений лицензионных программ (patch, service-pack). Их своевременная установка - одно из основных средств защиты персонального компьютера (особенно это касается антивирусных программ). Легальные пользователи оперативно и бесплатно получают все выпущенные обновления.

· Законность и престиж. Покупая нелегальное программное обеспечение, вы нарушаете закон, так как приобретаете "ворованные" программы. Вы подвергаете себя и свой бизнес риску юридических санкций со стороны правообладателей. У организаций, использующих нелегальное программное обеспечение, возникают проблемы при проверках лицензионной чистоты программного обеспечения, которые периодически проводят правоохранительные органы. За нарушение авторских прав в ряде случаев предусмотрена не только административная, но и уголовная ответственность. Нарушение законодательства, защищающего авторское право, может негативно отразиться на репутации компании. Нелегальные копии программного обеспечения могут стать причиной несовместимости программ, которые в обычных условиях хорошо взаимодействуют друг с другом.

· В ногу с техническим прогрессом. Управление программным обеспечением поможет определить потребности компании в программном обеспечении, избежать использования

устаревших программ и будет способствовать правильному выбору технологии, которая позволит компании достичь поставленных целей и преуспеть в конкурентной борьбе.

- Профессиональные предпродажные консультации. Преимущества приобретения лицензионного программного обеспечения пользователи ощущают уже при его покупке. Продажу лицензионных продуктов осуществляют сотрудники компаний - авторизованных партнеров ведущих мировых производителей программного обеспечения, квалифицированные специалисты. Покупатель может рассчитывать на профессиональную консультацию по выбору оптимального решения для стоящих перед ним задач.

- Повышение функциональности. Если у вас возникнут пожелания к функциональности продукта, вы имеете возможность передать их разработчикам; ваши пожелания будут учтены при выпуске новых версий продукта.

Приобретая нелегальное программное обеспечение вы очень рискуете.

Административная ответственность за нарушение авторских прав. Согласно статьи 7.12 КоАП РФ 1, ввоз, продажа, сдача в прокат или иное незаконное использование экземпляров произведений или фонограмм в целях извлечения дохода в случаях, если экземпляры произведений или фонограмм являются контрафактными: влечет наложение административного штрафа: на юридических лиц - от 300 до 400 МРОТ с конфискацией контрафактных экземпляров, произведений и фонограмм, а также материалов и оборудования, используемых для их воспроизведения, и иных орудий совершения административного правонарушения.

Уголовная ответственность за нарушение авторских прав. Согласно статьи 146 УК РФ (часть 2), незаконное использование объектов авторского права или смежных прав, а равно приобретение, хранение, перевозка контрафактных экземпляров произведений или фонограмм в целях сбыта, совершенные в крупном размере, наказываются штрафом в размере от 200 до 400 МРОТ или в размере заработной платы или иного дохода осужденного за период от двух до четырех месяцев, либо обязательными работами на срок от 180 до 240 часов, либо лишением свободы на срок до двух лет.

При использовании нелегального, то есть измененной пиратами версии, программного продукта, могут возникнуть ряд проблем:

- Некорректная работа программы. Взломанная программа – это изменённая программа, после изменений не прошедшая цикл тестирования.

- Нестабильная работа компьютера в целом.

Проблемы с подключением периферии (неполный набор драйверов устройств).

- Отсутствие файла справки, документации, руководства.

- Невозможность установки обновлений.

- Отсутствие технической поддержки продукта со стороны разработчика.

- Опасность заражения компьютерными вирусами (от частичной потери данных до полной утраты содержимого жёсткого диска) или другими вредоносными программами.

Теория схем программ

Теория схем программ – один из центральных разделов теоретического программирования, связанный с изучением структурных свойств и преобразований программ. В нем объектами исследования являются упрощенные математические модели программы, в которых с той или иной детализацией отражено строение программы, взаимодействие составляющих ее компонентов. Например, в операторных схемах программ такие понятия, как оператор, операнд, переменная, выполнение и т.д., являются обобщением соответствующих понятий существующих языков программирования. Каждая операторная схема программы моделирует целый класс конкретных программ, имеющих одинаковую информационно–логическую структуру, т.е. описывает его таким образом, что множество преобразований, корректных для схемы, образуется из преобразований, корректных для всех моделируемых ею программ. При переходе от программы к моделирующей ее схеме игнорируются несущественные особенности (например, синтаксические) конкретного языка программирования и сохраняются только те семантические свойства программы, которые используются при ее анализе и преобразовании.

Семантическая теория программ

Описание семантики языка в большинстве случаев осуществляется приведением нескольких простых примеров и кратким поясняющим текстом. Смысл этого текста чаще всего неоднозначен, так что различные читатели могут понимать его по-разному. Программист может получить ошибочное представление о том, что именно будет делать написанная им программа при выполнении, а разработчик может реализовать какую-либо языковую конструкцию иначе, чем разработчики других реализаций того же языка. Как и для синтаксиса, нужен какой-то метод, позволяющий дать удобочитаемое, точное и лаконичное определение семантики языка.

Задача определения семантики языка программирования рассматривается теоретиками давно, но до сих пор не найдено удовлетворительного универсального решения. Было разработано множество различных методов формального определения семантики.

Операционная семантика

Операционная семантика сводится к описанию смысла программы посредством выполнения ее конструкций на реальной или виртуальной вычислительной машине. Смысл конструкции определяется изменениями, произошедшими в состоянии машины после выполнения данной конструкции (оператора).

Допустим, что состояние компьютера – это значения всех его регистров и ячеек памяти, в том числе коды условий и регистры состояний. Если просто записать состояние компьютера, выполнить машинную команду, смысл которой нужно определить, затем записать новое состояние машины, а затем сравнить два полученных состояния, то семантика выполняемой команды станет понятной: она представляется изменением в состоянии машины, вызванным выполнением команды.

Описание смысла операторов языков программирования высокого уровня с помощью операционной семантики требует создания реального или виртуального компьютера. Аппаратное обеспечение компьютера является чистым интерпретатором его машинного языка. Интерпретатор любого языка программирования можно разработать с помощью программных инструментов, которые для данного языка становятся виртуальной машиной. Будем считать такой интерпретатор «чистым». Семантику языка программирования высокого уровня можно описать, используя «чистый» интерпретатор данного языка. Однако налицо две проблемы. Во-первых, сложность и индивидуальные особенности аппаратного обеспечения и операционного окружения, используемых для запуска чистого интерпретатора, затрудняют понимание выполняемых действий. Во-вторых, выполненное таким образом семантическое описание будет доступно только для пользователей с абсолютно идентичной конфигурацией вычислительной машины и одинаковой операционной системой.

Аксиоматическая семантика

Данная разновидность семантики была создана при разработке методов доказательства правильности программ, для которого применяется исчисление предикатов. Семантика каждой отдельной синтаксической конструкции языка определяется как набор аксиом или правил вывода. Этот набор используется для вывода результатов выполнения конструкции. Для описания смысла всей совокупности конструкций, образующих программу эти аксиомы и правила вывода следует использовать так же, как при доказательстве теорем в различных разделах математики. Если предположить, что значения входных переменных удовлетворяют некоторым ограничениям, то аксиомы и правила вывода могут быть использованы для получения ограничений на значения других переменных после выполнения каждой программной конструкции, например оператора. В итоге, когда программа завершена, будем иметь доказательство того, что полученные результаты удовлетворяют необходимым ограничениям на значения относительно входных значений. Иными словами, выходные данные представляют собой значения соответствующей функции, вычисленной по значениям входных данных.

Таким образом показывается, что программа выполняет именно то, что описано в ее спецификации. В доказательстве каждая конструкция (оператор) программы сопровождается предшествующим и последующим логическими выражениями, которые устанавливают ограничения на значения переменных, используемых в программе. В отличие от операционной семантики, требующей полного описания состояния абстрактной машины, для определения смысла конструкции в аксиоматической семантике используются только эти выражения.

Денотационная семантика

Это наиболее строгий из известных методов описания значения программ. Базисом денотационной семантики является теория рекурсивных функций.

Основная концепция – определение для каждой сущности языка некоторого математического объекта и функции, отображающей экземпляры синтаксической сущности в экземпляры математического объекта. Поскольку эти объекты строго определены, то они представляют собой точный смысл соответствующих им сущностей. Главная идея заключается в факте существования строгих методов работы с математическими объектами, а не имеющимися в языках программирования конструкциями. Сложность использования денотационной семантики заключается в создании объектов и функций отображения.

Модели вычислительных процессов: Модель графов распределения ресурсов

Модель вычислительного процесса

Абстрактно модель вычислительного процесса можно представить в виде схемы черного ящика.

Определим вычисления, определяющие процесс **P** и протекающие в вычислительных ресурсах **R** как обработку операционного пакета **I_{in}**, в результате чего формируется выходной пакет **I_{out}**. Пусть момент начала вычислений определяется входным управляющим воздействием **C_{in}**, а их завершение фиксируется по выдаче вычислительным ресурсом управляющего сигнала **C_{out}**. Графическое обозначение этих вычислений представлено на рисунке 1. Иерархическая организация вычислений позволяет говорить о том, что процесс можно разделить на более мелкие подпроцессы, каждый из которых может выполняться в отведенной для этого части общих ресурсов. При этом, одни и те же ресурсы могут повторно использоваться при выполнении различных вычислений, разделяясь подпроцессами во времени.

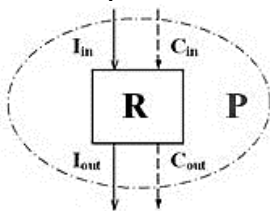


Рисунок 1 - Процесс как вычисления, протекающие в ресурсе

В данном случае ключевыми абстракциями, важными факторами, являются исходные данные – входной пакет **I_{in}** и входное управляющее воздействие **C_{in}**. В результате работы вычислительного процесса на выходе получится выходной пакет **I_{out}** и входное управляющее воздействие **C_{out}**.

Данная модель предусматривает, за счет наличия входного управляющего воздействия, влияние многих факторов – условий. К таким условиям можно условно отнести:

1. Условие готовности данных (**Information** - условие). Перед запуском процесса на его информационных входах должны присутствовать все необходимые аргументы и функции, составляющие операционный пакет. Отсутствие каких-либо данных к моменту запуска процесса ведет к получению неправильного результата. То есть, если существуют процессы, результаты которых являются аргументами рассматриваемого процесса, то они должны завершиться к моменту запуска текущего процесса.

2. Условие выделения ресурсов (**Resource** - условие). Выполнение процесса протекает в соответствующих ресурсах, поэтому, перед его запуском, эти ресурсы должны быть определены и предоставлены. Ресурсный вход должен содержать информацию, подтверждающую выделение ресурсов, необходимых для успешного выполнения процесса. В противном случае его запуск просто невозможен.

3. Условия подтверждения использования результатов (**Acknowledge** - условие). Ресурсы, занимаемые процессом, могут освобождаться или повторно использоваться только после того, как результаты вычислений будут использованы всеми процессами, являющихся приемниками этих результатов в качестве аргументов. В противном случае данные будут потеряны, что приведет к неправильным вычислениям. Необходимость задержки в освобождении ресурсов мотивируется

тем, что память, используемая для хранения операндов, является разделяемым ресурсом. В нее записываются результаты вычислений элементарного процесса. Из нее же происходит чтение аргументов, необходимых другим процессам.

Информация о выполнении условий готовности в рассматриваемой модели подтверждается наборами соответствующих управляющих сигналов:

1. сигналами готовности элементов операционного пакета;
2. сигналами готовности необходимых вычислительных ресурсов;
3. сигналами, подтверждающими отсутствие конфликтов при использовании разделяемых ресурсов.

Эти входные сигналы формируются на основе выходных сигналов завершающихся процессов. Завершение элементарного вычислительного процесса одновременно подтверждает готовность результатов и использование им, в качестве своих аргументов, результатов, полученных ранее в процессах - <передатчиках>. Естественно, что при более сложном представлении процессов, схема их взаимодействия и моменты формирования управляющих сигналов тоже будут сложнее.

Модель графов распределения ресурсов.

Объясним понятие «Тупики»

Для описания и исследования подобных ситуаций введем формальную модель системы в общем виде. С помощью модели будем представлять информацию о запросах процессов к ресурсам, о фактическом владении процессов ресурсами и об освобождении ресурсов.

Пусть в системе имеется m видов ресурсов (например, процессор, память, устройства ввода-вывода). Будем обозначать типы ресурсов в системе R_1, R_2, \dots, R_m . Пусть каждый тип ресурса R_i имеет W_i экземпляров. Каждый процесс может использовать ресурс одним из следующих способов:

- запрос (request);
- использование (use);
- освобождение (release).

Тупик может возникнуть, если одновременно выполняются следующие четыре условия:

- взаимное исключение: только один процесс в каждый момент времени может получить доступ к ресурсу;
- удержание и ожидание: процесс, удерживающий один ресурс, ожидает приобретения других ресурсов, которыми обладают другие процессы;
- отсутствие прерываний: процесс может освободить ресурс только добровольно, когда завершит свою работу;
- циклическое ожидание: существует множество $\{P_0, P_1, \dots, P_n\}$, такое, что P_0 ожидает ресурса, которым обладает P_1 ; P_1 ожидает ресурса, которым обладает P_2 ... P_n ожидает ресурса, которым обладает P_0 .

Граф распределения ресурсов

Множество вершин V и множество дуг E .

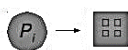
V подразделяется на два типа вершин:

- $P = \{P_1, P_2, \dots, P_n\}$, множество всех процессов в системе,
- $R = \{R_1, R_2, \dots, R_m\}$, множество всех ресурсов в системе,

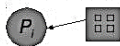
- Дуга типа "запрос" (request edge) – направленная дуга $P_i \rightarrow R_j$
- Дуга типа "присваивание" (assignment edge) – направленная дуга $R_j \rightarrow P_i$

Смысл различных направленностей дуг в следующем. Если процесс претендует на какой-либо ресурс, то дуга проводится из вершины процесса в вершину ресурса. Когда же конкретная единица ресурса уже выделена какому-либо конкретному процессу, то дуга, в знак этой принадлежности проводится из вершины ресурса в вершину процесса.

- P_i запрашивает экземпляр ресурса R_j



- P_i удерживает экземпляр ресурса R_j

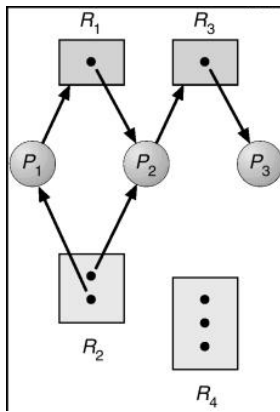


- Процесс



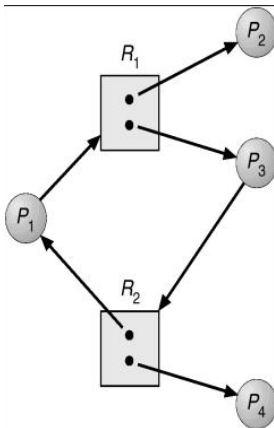
- Тип ресурса, имеющий 4 экземпляра





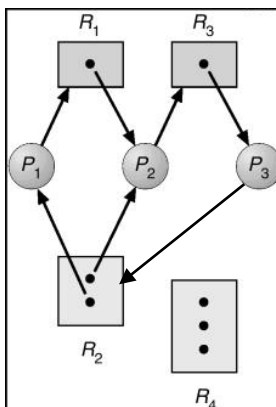
Граф распределения ресурсов

Данный граф изображает систему с 3 процессами и 4 видами ресурсов: ресурсы видов 1 и 3 имеют по 1 экземпляру, ресурс вида 2 – 2 экземпляра, ресурс вида 4 – 3 экземпляра. Процесс 1 претендует на ресурс 1, который занят процессом 2. Процесс 2 претендует на ресурс 3, который занят процессом 3. Две единицы ресурса 2 отданы процессам 1 и 2 соответственно. Ресурс 4 не распределялся (все три единицы свободны).



Граф распределения ресурсов с тупиком

Имеется ситуация циклического ожидания между процессами 1, 2 и 3. Процесс 1 претендует на ресурс, которым владеет процесс 2. Процесс 2 претендует на ресурс, которым владеет процесс 3. Процесс 3 претендует на ресурс, одна единица которого отдана процессу 1, а вторая – процессу 2.



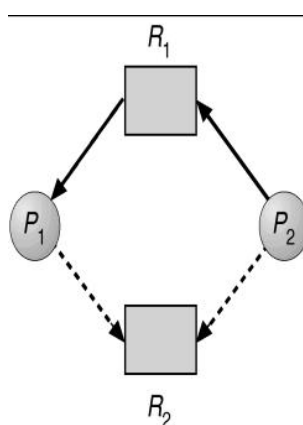
Граф распределения ресурсов с циклом, но без тупика

В данном случае (имеется четыре процесса и два вида ресурсов. В цикле участвуют вершины-процессы 1 и 3. Однако, благодаря тому, что каждого ресурса имеется по две единицы, тупика удастся избежать: процесс 1, ожидающей ресурса 1, сможет его получить, когда завершится процесс 2 (а не процесс 1), обладающей одной единицей данного ресурса и не входящий в цикл ожидания. Аналогично, процесс 3, претендующий на ресурс 2, сможет его получить после его освобождения процессом 4 (а не 1).

Если граф распределения ресурсов не содержит циклов, то в системе тупиков нет;

Если граф распределения ресурсов содержит цикл, то возможно два случая:

- если ресурсов каждого вида имеется только по одному экземпляру, то имеет место тупик;
- если ресурсов по несколько экземпляров, то тупик возможен.



Граф распределения ресурсов для стратегии избегания тупиков(ИТ)

Для реализации стратегии ИТ к данному графу необходимо добавить информацию не только о фактических, но и о возможных в будущем запросах ресурсов со стороны процессов. Для этого, в дополнение к дугам запросов и присваиваний, введем в рассмотрение дугу потребности (claim edge), которая ведет из вершины-процесса P_1 , в вершину-ресурс R_2 обозначается пунктирной линией и означает, что процесс P_1 может потреблять ресурс R_2 , когда процесс фактически запрашивает данный ресурс, дуга потребности преобразуется в дугу запроса (пунктирная линия → сплошной). Когда процесс освобождает ресурс, дуга присваивания преобразуется обратно в дугу потребности. Цель данной модификации графа – обеспечить, чтобы потребность в ресурсах была априорно известна системе.

Вычислительные схемы

Вычислительная схема - это представление в графической форме асинхронной системы, состоящей из набора операторов (процессов), которые воздействуют на множество «регистров» (данных). Каждая вычислительная схема определяется с помощью двух графов: графа потока данных и графа управления.

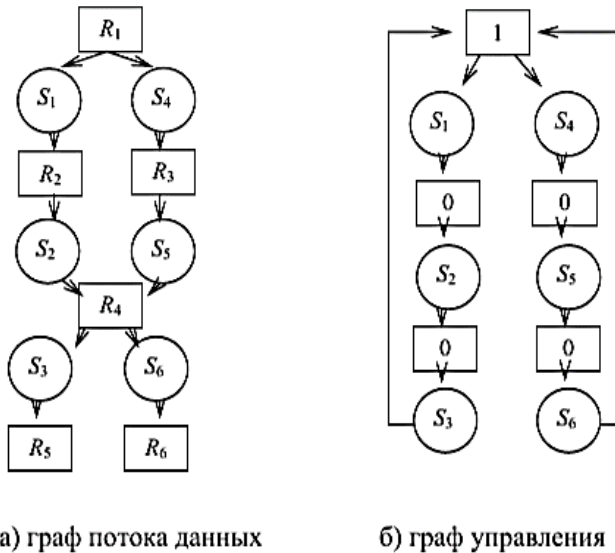


Рисунок 5.1 - Пример вычислительной схемы

Граф потока данных (информационный граф) определяет входные и выходные данные для каждого оператора. Дуга ($R_i S_k$) от регистра R_i к оператору S_k означает, что данные R_i являются элементом входных данных этого оператора;

дуга ($S_k R_j$) определяет данные R_j как выходные для S_k . Некоторые данные R могут являться выходными для оператора S_i и входными для оператора S_j . Пример графа потока данных для некоторой вычислительной схемы представлен на рисунке 5.1 а);

операторы и регистры данных представлены соответственно кружками и прямоугольниками.

Граф управления определяет последовательность выполнения операторов.

Каждый оператор (представлен кружком) связан с некоторым количеством управляющих счетчиков (представлены прямоугольниками). Каждый из управляющих счетчиков содержит неотрицательное целое число. Текущие значения счетчиков совместно со значениями данных на графе потока данных определяют состояние вычислительной схемы.

Пример графа управления представлен на рисунке 5.1 б).

Последовательность операторов $S_1, S_2, \dots, S_n, \dots$, называется **последовательностью исполнения схемы**.

Вычислительные схемы не являются конструктивной моделью, несмотря на свою интуитивную понятность и возможность сделать вывод о возникновении тупиков в системе.

Реляционные системы управления базами данных Объектно-ориентированные базы данных

Реляционные СУБД

СУБД – это программы, которые позволяют пользователям взаимодействовать с БД. СУБД позволяет управлять доступом к базе данных, записывать данные, отправлять запросы и выполнять любые другие задачи, связанные с управлением БД.

Однако для решения любой из этих задач СУБД должна иметь какую-то базовую модель, которая определяет, как организованы данные. Реляционная модель – это один из подходов к организации данных, который появился в конце 1960-х.

История реляционной модели

БД – это логически смоделированные кластеры информации. Любая коллекция данных является базой данных, независимо от того, как и где она хранится. Даже физическая папка, содержащая информацию о заработной плате, является базой данных, как и стопка больничных бланков пациентов.

Одной из первых моделей БД была иерархическая модель, данные в которой организованы в древовидную структуру, аналогичную современным файловым системам.

Иерархическая модель широко применялась в ранних СУБД, однако оказалась недостаточно гибкой. Отдельные записи в ней могут иметь несколько дочерних записей, однако по иерархии каждая запись может иметь только одного «родителя». Поэтому ранние иерархические базы данных были ограничены только отношениями «один к одному» и «один ко многим». Отсутствие поддержки отношения «многие ко многим» становилось проблемой при работе с точками данных, которые нужно связать с несколькими родителями.

В конце 1960-х ученый-компьютерщик Эдгар Ф. Кодд, работавший в IBM, разработал реляционную модель управления базами данных. Реляционная модель Кодда позволяет связывать отдельные записи более чем с одной таблицей, тем самым создавая между точками данных отношения «многие ко многим» (в дополнение к отношениям «один ко многим»). Когда дело касалось проектирования структур БД, эта модель обеспечила большую гибкость, чем другие существующие на тот момент модели. Это означало, что реляционные системы управления базами данных (РСУБД) могли удовлетворить гораздо более широкий спектр потребностей.

Кодд предложил язык для управления реляционными данными по имени Alpha, который повлиял на развитие более поздних языков БД. Двое коллег Кодда из IBM, Дональд Чемберлин и РэймондБойс, создали свой язык, вдохновленный Alpha. Они назвали его SEQUEL (StructuredEnglishQueryLanguage), но такая торговая марка уже существовала, и тогда они сократили название языка до SQL (StructuredQueryLanguage).

Из-за аппаратных ограничений ранние реляционные базы данных были очень медленными. Чтобы технология получила широкое распространение, потребовалось некоторое время. Но к середине 1980-х годов реляционная модель Кодда была реализована в ряде коммерческих продуктов для управления базами данных как от IBM, так и от ее конкурентов. Эти вендоры последовали примеру IBM, разработав и внедрив свои собственные диалекты SQL. К 1987 году и Американский национальный институт стандартов, и Международная организация по стандартизации ратифицировали и опубликовали стандарты для SQL, укрепив его статус как принятого языка для управления СУБД.

Благодаря такому широкому использованию реляционной модели во многих отраслях она стала стандартной моделью для управления данными. Даже с появлением баз данных NoSQL реляционные БД остаются доминирующими инструментами для хранения и организации данных.

В связи с резким ростом популярности РСУБД в 1980-х годах многие компании стали позиционировать свои СУБД как «реляционные» в рекламных целях, иногда не имея для этого достаточных оснований, вследствие чего автор реляционной модели данных Эдгар Кодд в 1985 году опубликовал свои знаменитые «12 правил Кодда», которым должна удовлетворять каждая РСУБД.

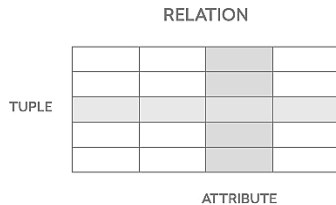
13 правил (в данном случае исчисление начинается с 0), которым должна удовлетворять каждая РСУБД .

В действительности правила столь строги, что все популярные так называемые «реляционные» СУБД не соответствуют многим критериям.

Как реляционные БД организуют (структурируют) данные

Теперь, когда у вас есть общее понимание истории реляционной модели, давайте более подробно рассмотрим, как данная модель структурирует данные.

Наиболее значимыми элементами реляционной модели являются *отношения*, которые известны пользователям и современным РСУБД как *таблицы*. Отношения — это набор *кортежей*, или строк в таблице, где каждый кортеж имеет набор *атрибутов*, или столбцов:

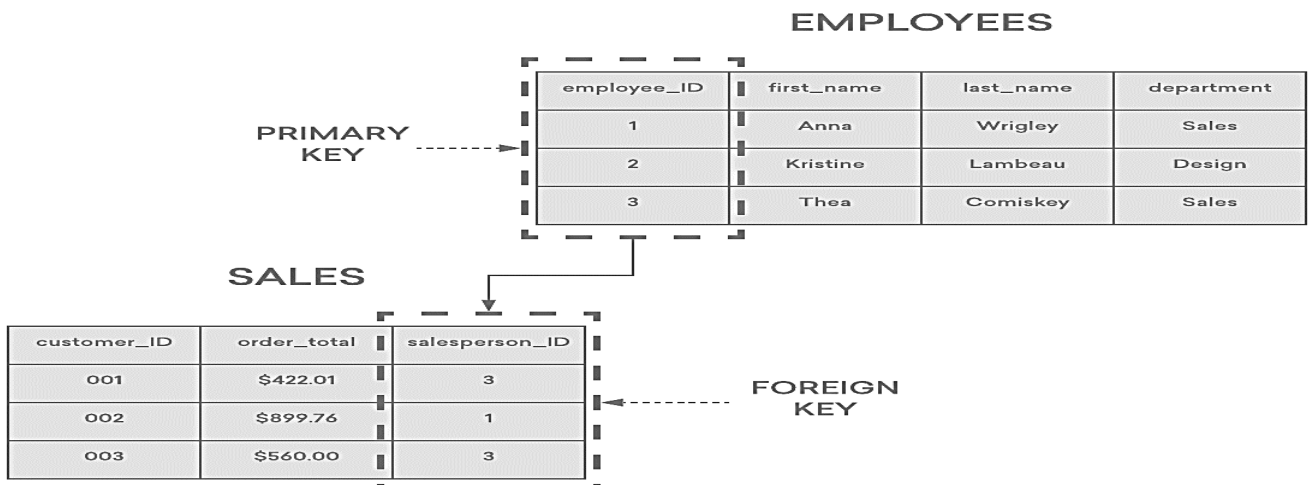


Столбец — это наименьшая организационная структура реляционной базы данных, представляющая различные ячейки, которые определяют записи в таблице. Отсюда происходит более формальное название — атрибуты. Вы можете рассматривать каждый кортеж в качестве уникального экземпляра чего-либо, что может находиться в таблице: категории людей, предметов, событий или ассоциаций. Такими экземплярами могут быть сотрудники компаний, продажи в онлайн-бизнесе или результаты лабораторных тестов. Например, в таблице с трудовыми записями учителей в школе кортежи могут иметь такие атрибуты, как `name`, `subjects`, `start_date` и т.д.

При создании столбцов вы указываете *тип данных*, определяющий, какие записи могут вноситься в данный столбец. РСУБД часто используют свои собственные уникальные типы данных, которые могут не быть напрямую взаимозаменяемы с аналогичными типами данных из других систем. Некоторые распространенные типы данных включают даты, строки, целые числа и логические значения.

В реляционной модели каждая таблица содержит по крайней мере один столбец, который можно использовать для уникальной идентификации каждой строки. Он называется *первичным ключом*. Это важно, поскольку это означает, что пользователям не нужно знать, где физически хранятся данные на компьютере. Их СУБД может отслеживать каждую запись и возвращать ее в зависимости от конкретной цели. В свою очередь, это означает, что записи не имеют определенного логического порядка, и пользователи могут возвращать данные в любом порядке или с помощью любого фильтра по своему усмотрению.

Если у вас есть две таблицы, которые вы хотите связать друг с другом, можно сделать это с помощью *внешнего ключа*. Внешний ключ — это, по сути, копия основного ключа одной таблицы (таблицы «предка»), вставленная в столбец другой таблицы («потомка»). Следующий пример показывает отношения между двумя таблицами: одна используется для записи информации о сотрудниках компании, а другая — для отслеживания продаж компании. В этом примере первичный ключ таблицы `EMPLOYEES` используется в качестве внешнего ключа таблицы `SALES`:



Если вы попытаетесь добавить запись в таблицу «потомок», и при этом значение, вводимое в столбец внешнего ключа, не существует в первичном ключе таблицы «предок», вставка будет недействительной. Это помогает поддерживать целостность уровня отношений, поскольку ряды в обеих таблицах всегда будут связаны корректно.

Структурные элементы реляционной модели помогают хранить данные в структурированном виде, но хранение имеет значение только в том случае, если вы можете извлечь эти данные. Для извлечения информации из РСУБД вы можете создать *запрос*, т. е. структурированный запрос на набор информации. Как уже упоминалось ранее, большинство реляционных баз данных используют язык SQL для управления данными и отправки запросов. SQL позволяет фильтровать результаты и обрабатывать их с помощью различных пунктов, предикатов и выражений, позволяя вам контролировать, какие данные появятся в результате.

Преимущества и недостатки реляционных баз данных

Ознакомившись с организационной структурой реляционных баз данных, давайте теперь рассмотрим некоторые из их преимуществ и недостатков.

Современный SQL и базы данных, реализующие его, несколько отличаются от реляционной модели Кодда. Например, модель Кодда диктует, что каждая строка в таблице должна быть уникальной, в то время как из соображений практичности большинство современных РБД допускают дублирование строк. Некоторые пользователи не считают базу данных SQL «настоящей» реляционной базой, если она не соответствует каждой из спецификаций реляционной модели, описанной Коддом. Однако на практике любая СУБД, использующая SQL и хотя бы в некоторой степени придерживающаяся реляционной модели, скорее всего, будет относиться к РСУБД.

Популярность реляционных баз данных быстро росла, а вместе с этим росла и ценность данных. В связи с этим начали проявляться некоторые недостатки реляционной модели. Во-первых, реляционную базу данных сложно масштабировать по горизонтали. (*Горизонтальное масштабирование* – это практика добавления новых машин к существующему стеку, чтобы распределить нагрузку и обеспечить более быструю обработку трафика).

Примечание: Горизонтальное масштабирование часто противопоставляется *вертикальному*, которое подразумевает обновление оборудования существующего сервера (обычно путем добавления дополнительной оперативной памяти или процессора).

Причина, по которой реляционную базу данных сложно масштабировать по горизонтали, связана с тем фактом, что *реляционная модель предназначена для обеспечения согласованности*. Следовательно, клиенты, запрашивающие одну и ту же базу данных, всегда будут получать одни и те же данные. Но если реляционную базу данных масштабировать по горизонтали и разместить на нескольких машинах, ей становится сложно обеспечить согласованность, поскольку клиенты могут записывать данные на одну ноду, но не на другие. Вероятно, между внесением записи и ее отражением на других нодах пройдет некоторое время, а подобная задержка приведет к несогласованности данных.

Еще одно ограничение, представленное РСУБД, заключается в том, что реляционная модель была разработана для управления структурированными данными (то есть данными, которые соответствуют предопределенному типу или, по крайней мере, организованы некоторым заранее определенным образом, благодаря чему их легко сортировать и искать). Однако с распространением персональных компьютеров и появлением Интернета в начале 1990-х годов широкое распространение получили *неструктурированные данные* (сообщения электронной почты, фотографии, видео и т.п.).

Еще одно преимущество реляционных баз данных состоит в том, что почти каждая СУБД поддерживает транзакции. Транзакция состоит из одного или нескольких отдельных SQL-операторов, выполняемых последовательно как единая задача. Транзакции представляют собой подход «все или ничего»: каждый SQL-оператор в транзакции должен быть действительным; в противном случае вся транзакция не будет выполнена. Это обеспечивает целостность данных при внесении изменений в несколько строк или таблиц.

SQL также является чрезвычайно мощным инструментом, он позволяет добавлять и изменять данные на ходу, а также менять структуру схем и таблиц БД, не влияя на существующие данные.

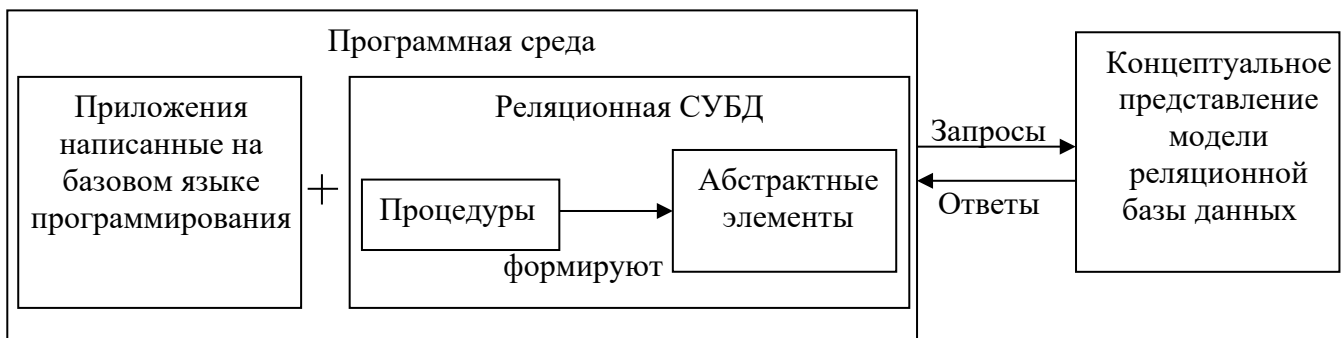
Концептуальное представление модели реляционной базы данных — это набор таблиц, состоящих из строк и столбцов. (Например, информацию о сотрудниках компании можно просматривать в таблице, где для каждого сотрудника отведена одна строка, а в столбцах содержатся имя (Name), адрес (Address), идентификационный номер (Emplld) и прочие данные.)

Реляционная СУБД включает процедуры, которые позволяют приложению выбирать определенные элементы определенной строки таблицы или, к примеру, выводить диапазон значений, найденных в столбце (зарплаты).

Эти процедуры формируют абстрактные инструменты, которые используются приложениями для доступа к базе данных.

Приложение обычно пишется на одном из языков программирования общего назначения. В этих языках обычно не хватает инструментов для обращения к базам данных. Процедуры СУБД расширяют возможности используемого языка, в том смысле, что поддерживают концептуальный образ модели базы данных.

Исходный язык общего назначения расширяется процедурами СУБД, (поэтому его часто называют базовым языком (hostlanguage)). Таким образом, базовый язык, расширенный при помощи СУБД, является программной средой для разработки приложений работающих с базами данных, и при помощи этой программной среды с базой можно работать так, как если бы она была организована согласно концептуальной модели.



СУБД на инвертированных файлах

Организация доступа к данным на основе инвертированных списков используется практически во всех современных реляционных СУБД, но в этих системах пользователи не имеют непосредственного доступа к инвертированным спискам (индексам).

База данных, организованная с помощью инвертированных списков, похожа на реляционную БД, но с тем отличием, что хранимые таблицы и пути доступа к ним видны пользователям. При этом:

1. Строки таблиц упорядочены системой в некоторой физической последовательности.
2. Физическая упорядоченность строк всех таблиц может определяться и для всей БД.
3. Для каждой таблицы можно определить произвольное число ключей поиска, для которых строятся индексы. Эти индексы автоматически поддерживаются системой, но явно видны пользователям.

Поддерживаются два класса операторов:

1. Операторы, устанавливающие адрес записи, среди которых:
 - прямые поисковые операторы;
 - операторы, находящие запись в терминах относительной позиции от предыдущей записи по некоторому пути доступа.
2. Операторы над адресуемыми записями

Общие правила определения целостности БД отсутствуют. В некоторых системах поддерживаются ограничения уникальности значений некоторых полей, но в основном все возлагается на прикладную программу.

Однако такие СУБД обладают рядом ограничений на количество файлов для хранения данных, количество связей между ними, длину записи и количество ее полей.

<<В основном файле разрешается выделить один или несколько атрибутов (выделяемый атрибут может быть как первичным, так и вторичным ключом), по значениям которых затем будут формироваться инвертированные файлы и списки связи.

Все записи файлов получают в пределах БД единую нумерацию. Каждому значению ключевого атрибута ставится в соответствие множество номеров записей основных файлов, где это значение связано с именем атрибута.

Определенная таким образом последовательность значений атрибута А и номеров записей основного файла является инвертированным файлом.

Единая нумерация всех записей базы данных приводит к тому, что номер записи становится первичным ключом во всех основных файлах базы данных независимо от того, какие атрибуты образуют ключ в каждом из этих файлов.

Для двух файлов, имеющих общий атрибут, существуют два списка связи. В первом списке для каждого номера записи из первого файла указываются номера записей из второго файла, имеющие то же самое значение атрибута. Аналогично определяется содержимое второго списка связи.>>

Гипертекстовые и мультимедийные БД

Суть гипертекстовой технологии заключается в том, что текст представляется как многомерный, т.е. с иерархической структурой типа сети. Материал текста делится на фрагменты. Каждый видимый на экране ЭВМ фрагмент, дополненный многочисленными связями с другими фрагментами, позволяет уточнить информацию об изучаемом объекте и двигаться в одном или нескольких направлениях по выбранной связи.

Под гипертекстом понимают систему информационных объектов (статей), объединенных между собой направленными связями, образующими сеть. Каждый объект связывается с информационной панелью экрана, на которой пользователь может выбирать одну из связей. Объекты могут быть текстовыми, графическими, музыкальными, с использованием средств мультимедиа, аудио- и видеотехники. Вместо традиционных методов поиска информации по соответствующему поисковому ключу гипертекстовая технология предполагает перемещение от одних объектов информации к другим с учетом их смысловой, семантической связанности.

Структурно гипертекст состоит из информационного материала, тезауруса гипертекста, списка главных тем и алфавитного словаря.

Информационный материал подразделяется на информационные статьи, состоящие из заголовка статьи и текста. Заголовок содержит тему или наименование описываемого объекта. Информационная статья содержит традиционные определения и понятия, должна занимать одну панель и быть легко обозримой.

Тезаурус (лат. сокровище, запас, богатство) гипертекста - это автоматизированный словарь, отображающий семантические отношения между лексическими единицами дескрипторного информационно-поискового языка и предназначенный для поиска слов по их смысловому содержанию. Он состоит из тезаурусных статей, которые имеют заголовки и список заголовков родственных тезаурусных статей. Заголовок тезаурусной статьи совпадает с наименованием информационной статьи и является наименованием объекта, описание которого содержится в информационной статье.

Список главных тем содержит заголовки всех справочных статей, для которых нет ссылок типа род - вид, часть - целое.

Алфавитный словарь включает в себя перечень наименований всех информационных статей в алфавитном порядке.

Гипертекстовая технология применяется в издательской деятельности, библиотечной работе, обучающих системах, при разработке документации, законов, справочных руководств, баз данных, баз знаний и т. д.

Базы данных, содержащие мультимедийную информацию, теория относится к базам данных пятого (последнего) поколения. Они получили название мультимедийных баз данных (ММБД) (другое название – мультисредные). Особенности ММБД, вызванные структурной сложностью и неоднородностью хранимой в них информации, показывают, что построение

систем баз данных пятого поколения является достаточно сложной задачей, которую преждевременно считать решенной.

Системы управления базами данных (СУБД), используемые в настоящее время в АСУ, относятся к классу реляционных СУБД (РСУБД). Многие РСУБД обладают возможностью ранения в составе своих таблиц полей с мультимедиа данными. Однако традиционные операции по манипулированию данными, применимые к элементарным данным, в отношении мультимедиа данных в РСУБД не поддерживаются.

По этой причине прямое использование РСУБД в качестве СУ ММБД недостаточно и невозможно, что позволяет определить основную проблему построения ММБД в АСУ как несоответствие используемых в АСУ программно-инструментальных средств управления базами данных потребностям обработки в АСУ мультимедийной информации.

Для решения данной проблемы необходимо использование СУБД, поддерживающих объектно-ориентированную (объектную) модель данных. При этом первоочередной задачей следует считать разработку концептуальной модели (КМ) ММБД, которая модель затем преобразуется в объектно-ориентированную модель ММБД логического уровня.

Таким образом, применение разработанной концептуальной модели в интересах объектно-ориентированной реализации ММБД связывается с ее преобразованием в объектно-ориентированную логическую структуру БД в соответствии со специально разработанным для этой цели алгоритмом. Саму разработку такого алгоритма следует считать направлением дальнейших исследований по данной тематике.

Одним из перспективных направлений развития гипертекстовых систем является технология гипермедиа — соединение технологии гипертекста и технологии мультимедиа (интеграция текста, графики, звука, видео).

XML-серверы

По существу, XML сервер — это сервер, на котором хранятся и с которого можно скачать документы в формате XML.

Используя технологии XML, можно производить обмен данными между различными приложениями на различных платформах. Поэтому организации переходят на хранение своих данных в формате XML, чтобы пользоваться преимуществами XML-технологий. Хранение данных в этом формате улучшает доставку и масштабируемость Ваших приложений при низких управленческих и эксплуатационных расходах.

В традиционных реляционных базах данных данные хранятся в виде строк и столбцов, что может быть слишком сложным. Но в случае XML сервера эта сложность устранена и здесь можно хранить любые данные, включая мультимедийные файлы и даже реляционные данные. XML-сервер — это любой сервер, выходным продуктом для которого является документ XML, который может использоваться другими приложениями для обработки.

Компонент <<XML-сервер>>, представляет собой промежуточный уровень между уровнем приложения и уровнем СУБД.

XML-сервер взаимодействует с приложением на основе XML-интерфейса, представляющего собой входные и выходные потоки данных в формате XML.

XML-сервер и СУБД взаимодействуют на основе стандартных интерфейсов доступа к реляционным базам данных, таких как ODBC или JDBC

Задачи, решаемые XML-сервером:

- организация удобного для программистов XML-интерфейса для доступа к СУБД (отпадает необходимость низкоуровневого взаимодействия с СУБД)
- поддержка распределенных и разнородных баз данных (так как каждая СУБД обладает присущими только ей характеристиками, возникает проблема поддержки распределенных и разнородных баз данных. XML Server может учитывать конкретные реализации СУБД, обеспечивая “прозрачность” информационной системы для разработчиков)
- промежуточное хранение данных (однократно запускаемые JAVA и CGI процессы не могут хранить промежуточные данные в ОП. Соответственно XML server может устранить данный недостаток)

- кэширование данных (повторные данные берутся непосредственно из XML Server без обращения к СУБД)
- индексирование данных (частичный выбор данных, определенных пользователем)

Объектно-ориентированные базы данных

Одной из новейших областей исследований баз данных является их создание на основе объектно-ориентированной парадигмы. В результате получается объектно-ориентированная база данных (object-oriented database), состоящая из объектов, связи между которыми отражают отношения между объектами. Например, объектно-ориентированная реализация базы данных сотрудников:

Отношение EMPLOYEE			
Empl Id	Name	Address	SSN
25X15	Джо Бейкер	ул. Новая 33	111223333
34Y70	Шери Кларк	Верхнегородский пр. 563	999009999
23Y34	Джерри Смит	Круглый пер. 1555	111005555
⋮	⋮	⋮	⋮

Отношение JOB			
Job Id	Job Title	Skill Code	Dept
S25X	Секретарь	T5	Отдел кадров
S26Z	Секретарь	T6	Бухгалтерия
F5	Нач. группы	FM3	Отдел сбыта
⋮	⋮	⋮	⋮

Отношение ASSIGNMENT			
Empl Id	Job Id	Start Date	Term Date
23Y34	S25X	3-1-1999	4-30-2001
34Y70	F5	10-1-2001	*
23Y34	S25Z	5-1-2001	*
⋮	⋮	⋮	⋮

Рис. 9.5. База данных сотрудников, состоящая из трех отношений

могла бы включать три класса (то есть три типа объектов): EMPLOYEE, JOB и ASSIGNMENT. Объект класса EMPLOYEE мог бы содержать такие элементы, как EmplId, Name, Address и SSNum; объект класса JOB — элементы JobId, JobTitle, Skill Code и Dept; объект класса ASSIGNMENT — элементы StartDate и TermDate.

Концептуальное представление такой базы данных (рис. 11) образуется объектами и соединяющими их линиями, показывающими отношения между различными объектами. Рассмотрев объект типа EMPLOYEE, мы увидим, что он связан с набором объектов типа ASSIGNMENT, представляющих различные назначения сотрудника на должности. В свою очередь, каждый из объектов типа ASSIGNMENT связан с объектом типа JOB, обозначающим должность, которую занимал или занимает сотрудник. Таким образом, все назначения сотрудника на различные должности можно найти, проследив связи, идущие от объекта, представляющего сотрудника. Аналогично можно узнать, какие сотрудники занимали определенную должность, изучив ссылки от объекта, представляющего должность.

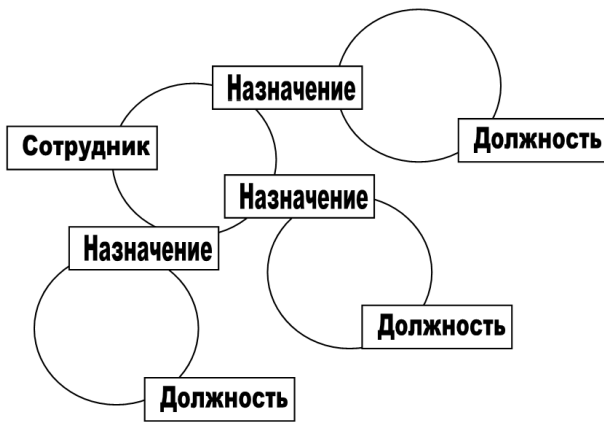


Рисунок 11 – Связи между объектами в объектно-ориентированной базе данных

Связи между объектами в объектно-ориентированной базе данных обычно поддерживаются СУБД, поэтому подробности их реализации не касаются программиста, разрабатывающего приложения. Когда новый объект добавляется в базу данных, приложению необходимо только указать, с какими объектами его нужно связать в базе. СУБД сама создает необходимую для регистрации этих связей систему указателей. В частности, СУБД может связать объекты, обозначающие назначения на разные должности определенного сотрудника, в стиле, схожем со связным списком.

Другая задача объектно-ориентированной СУБД — обеспечивать постоянное хранение переданных ей объектов. Такое требование может казаться очевидным, но в действительности хранение отличается от обычного способа работы с объектами. В обыкновенной объектно-ориентированной программе объекты, созданные в ходе выполнения, удаляются во время завершения программы. В этом смысле объекты считаются временными. Но объекты, созданные и помещенные в базу данных, должны быть постоянными — то есть их необходимо сохранить после того, как программа, создавшая их, завершится. Поэтому обеспечение постоянного хранения объектов — это существенное отклонение от нормы.

Сторонники объектно-ориентированных баз данных приводят множество аргументов в защиту того, что объектно-ориентированный подход к разработке базы данных лучше, чем реляционный. Один из аргументов заключается в том, что он обеспечивает единую парадигму разработки всей системы программного обеспечения (приложения, СУБД и самой базы данных). В противоположность этому, исторически приложения для обмена данными с реляционными базами данных создаются на императивном языке программирования. Но при этом всегда возникают конфликты между императивной и реляционной парадигмами. На нашем уровне изучения это различие едва ли заметно, но оно всегда являлось причиной множества ошибок в программном обеспечении. Однако мы можем оценить тот факт, что объектно-ориентированная база данных в сочетании с объектно-ориентированным приложением создают однородный образ объектов, общающихся между собой внутри единой системы, тогда как реляционная база данных и императивное приложение вызывают образ двух структур различной природы, пытающихся найти общий интерфейс.

Чтобы понять другое преимущество объектно-ориентированных баз данных над их реляционными конкурентами, рассмотрим проблему хранения имен сотрудников в реляционной базе данных. Если полное имя (фамилия, имя и отчество) целиком хранится в одном атрибуте отношения, то выполнять запросы, относящиеся только к фамилиям, становится неудобно. Если же имя хранится в трех разных атрибутах — фамилия, имя и отчество, — то вызывает неудобства обработки имен, не подходящих под шаблон «фамилия, имя, отчество». В объектно-ориентированной базе данных эти проблемы можно спрятать в объекте, где хранится имя сотрудника. Имя можно записать в интеллектуальный объект, который сможет выдать имя сотрудника в различных форматах. Так, с использованием объектов обработка только фамилий становится такой же простой, как и работа с полными именами, девичьими фамилиями или прозвищами. Детали, связанные с каждой задачей, будут инкапсулированы в пределах объектов.

Возможность инкапсулировать технические подробности различных форматов данных полезна и в других случаях. В реляционной базе данных атрибуты отношения являются частью общего дизайна базы данных, поэтому типы, связанные с этими атрибутами, появляются во всей

СУБД. (Необходимо объявлять временные переменные подходящих типов и разрабатывать процедуры для обработки данных различных типов.) Таким образом, расширение реляционной базы данных за счет добавления атрибутов новых типов (к примеру, звуковых и видео) может быть проблематичным. Действительно, это потребует расширения множества процедур во всей базе данных, чтобы они могли работать и с новыми типами. С другой стороны, в объектно-ориентированной модели процедуры, которые используются для получения объектов, обозначающих имена сотрудников, могут извлекать информацию, например из объекта, представляющего видеоклип, так как различие в типах может быть скрыто внутри соответствующих объектов. Следовательно, при помощи объектно-ориентированного подхода удобнее конструировать мультимедийные базы данных — эта возможность уже является существенным преимуществом.

Еще одно преимущество, которое объектно-ориентированная парадигма предлагает для разработки баз данных, — это возможность **хранения интеллектуальных объектов** вместо обычных данных. Это означает, что **объект может содержать методы, описывающие, как он будет отвечать на сообщения**, относящиеся к его содержимому и связям. Например, каждый объект класса EMPLOYEE (см. рис. 11) может содержать методы для вывода и обновления информации, хранящейся в объекте, а также для вывода предыдущих должностей сотрудника и, возможно, для регистрации перевода сотрудника на новую должность. Аналогично, каждый объект класса JOB может обладать методами для вывода характеристик должности и, возможно, для вывода списка сотрудников, которые занимали эту должность. Так, чтобы получить историю работы сотрудника в компании, нам не нужно будет создавать внешние процедуры, описывающие, как получить информацию. Вместо этого мы просто попросим соответствующий объект сотрудника рассказать его историю работы. Таким образом, способность создавать базы данных, объекты которых могут интеллектуально отвечать на запросы, предлагает исключительный набор возможностей, далеко превосходящий возможности традиционных реляционных баз данных.

В объектно-ориентированных базах данных, в отличие от реляционных, хранятся не записи, а объекты. ОО-подход представляет более совершенные средства для отображения реального мира, чем реляционная модель, естественное представление данных.

К сожалению, в объектно-ориентированном программировании отсутствуют общие средства манипулирования данными, такие как реляционная алгебра или реляционное счисление. Работа с данными ведется с помощью одного из объектно-ориентированных языков программирования общего назначения, обычно это SmallTalk, C++ или Java.

Подведем теперь некоторые итоги:

В реляционной модели все отношения принадлежат одному уровню, именно это осложняет преобразование иерархических связей модели «сущность-связь» в реляционную модель. ОО-модель можно рассматривать послойно, на разных уровнях абстракции.

Имеется возможность определения новых типов данных и операций с ними.

В то же время, **ОО-модели присущ и ряд недостатков:**

1. · отсутствуют мощные непроцедурные средства извлечения объектов из базы. Все запросы приходится писать на процедурных языках, проблема их оптимизации возлагается на программиста;

2. · вместо чисто декларативных ограничений целостности (типа явного объявления первичных и внешних ключей реляционных таблиц с помощью ключевых слов PRIMARY KEY и REFERENCES) или полудекларативных триггеров для обеспечения внутренней целостности приходится писать процедурный код.

Очевидно, что оба эти недостатка связаны с отсутствием развитых средств манипулирования данными. Эта задача решается двумя способами — расширение ОО-языков в сторону управления данными (стандарт ODMG), либо добавление объектных свойств в реляционные СУБД (SQL-3, а также так называемые объектно-реляционные СУБД).

Основными принципами объектно-ориентированной технологии являются:

- абстракция;
- инкапсуляция;
- модульность;

- иерархия;
- типизация;
- полиморфизм;
- наследование.

Объектно-ориентированной БД (ООБД) является БД, которая основывается на принципах объектноориентированной технологии.

Организация процессов обработки данных в БД

Обработка данных включает набор различных функций и операций, которые можно разделить на три группы:

- поиск, фильтрация и сортировка данных;
- запросы к базе данных;
- механизм реализации событий, правил (триггеров) и процедур в базе данных.

Отличительная особенность операций по поиску, фильтрации и сортировке данных заключается в том, что они осуществляются в режиме открытой таблицы или формы. Результатом операций по поиску или фильтрации данных является изменение состояния просмотра открытой таблицы (формы), но не самих данных, которые физически остаются в той же таблице и в том же порядке.

Поиск записи по ее номеру производится на основе механизма распределения записей по страницам файла данных. Результатом такого поиска является перевод табличного курсора в ключевое поле соответствующей записи или «показ» в форме полей искомой записи.

Поиск записи по значению поля осуществляется также на основе механизма распределения записей по страницам файла данных с использованием техники вхождения образца в значения просматриваемого поля. Результатом поиска является установка табличного курсора в соответствующее поле найденной записи. Если записей с искомым значением выделенного поля несколько, тогда, как правило, реализуется последовательная «остановка» (последовательный просмотр) табличного курсора в соответствующих полях найденных записей.

Фильтр представляет собой набор условий, применяемых для отбора подмножества записей. Результатом фильтрации является «показ» (отображение) в открытой таблице или форме только отфильтрованных записей с временным «скрытием» всех остальных записей.

Набор условий, определяющих фильтр, формируется в различных СУБД по-разному, но общепринятым является использование выражений в условиях отбора данных. Аргументами выражения могут быть числа, даты, текст, имена полей, которые соединяются знаками математических операций (+, -, *, /), неравенств (>, <, =) и логических операций (AND, OR, NOT). При этом текстовые значения и аргументы заключаются в кавычки («Иванов»), даты в символы # (#01.01.03#).

Упорядочивание записей по возрастанию/убыванию или по алфавиту по определенному полю реализуется сортировкой данных. При этом в файле базы данных строки таблицы физически остаются не упорядоченными. Сортировка строк открытой таблицы происходит только в буферах страниц в оперативной памяти. Новый порядок расположения строк таблицы может быть зафиксирован специальной командой при закрытии таблицы.

При больших объемах таблиц (при большом количестве строк-записей) операции сортировки могут занимать продолжительное время, которое существенно сокращается, если сортировка осуществляется по индексированному полю.

Запросы являются наиболее распространенным видом обработки данных. Запрос представляет собой спецификацию (предписание) на специальном языке (языке базы данных) для обработки данных. В реляционных СУБД запросы к базе данных выражаются на языке SQL.

Формирование запросов в СУБД может осуществляться в специальном редакторе (командный режим) или через наглядно-диалоговые средства (конструкторы) и пошаговые мастера формирования запросов. Сформированный запрос в виде SQL-инструкции сохраняется в файле базы данных и впоследствии специальной командой СУБД может запускаться (открываться) на выполнение.

С точки зрения решаемых информационных задач и формы результатов исполнения

запросов их можно разделить на три группы:

- запросы на выборку данных;
- запросы на изменение данных;
- управляющие запросы.

Запросы на выборку относятся к одному из наиболее часто применяемых видов запросов. Данный вид запросов реализуется SQL-инструкцией SELECT с предложением FROM. Результатом исполнения запроса на выборку является набор данных, который представляет временную таблицу данных со структурой (поля, их типы и параметры), определяемой параметрами запроса и параметрами полей таблиц, из которых выбираются данные. Результаты запросов на выборку помещаются в специальную временную таблицу, размещаемую на период исполнения («открытия») запроса в оперативной памяти.

Важное значение для решения различных технологических информационных задач по ведению базы данных имеют запросы на изменение данных. В отличие от непосредственного ввода данных в режимах открытой таблицы или формы они вносят изменения сразу в группу записей за одну операцию. Таким образом, результатом запросов на изменение является не набор данных, как в запросах на выборку, а изменение данных в самих таблицах.

При исполнении запроса на удаление за одну операцию осуществляется удаление группы записей из одной или нескольких таблиц. Запросы на удаление реализуются SQL-инструкцией DELETE:

DELETE (поля) FROM (таблица) WHERE (условие);

Запрос на обновление за одну операцию вносит общие изменения в группу записей одной или нескольких таблиц. Реализуются SQL-инструкцией UPDATE. Запросы на обновления применяются тогда, когда необходимо осуществить глобальные однотипные изменения в каком-либо наборе данных.

UPDATE (таблица) SET (поле = значение) WHERE(условие);

Запрос на добавление осуществляет добавление группы записей из одной или нескольких таблиц в конец другой или группы других таблиц. При этом количество и типы полей при вставке записей должны совпадать.

INTER INTO таблица SELECT поля FROM таблица WHERE (условие);

Запросы на создание таблицы за одну операцию создают новую таблицу с заполненными данными на основе всех или части данных из одной или нескольких таблиц.

SELECT поля INTO таблица FROM таблица WHERE (условие);

В составе языка описания данных DDL имеются ряд SQL-инструкций, на основе которых строятся запросы по созданию/модификации реляционных таблиц или отдельных их элементов. Такие запросы называются управляющими.

Запросы на создание таблицы реализуются SQL-инструкцией CREATETABLE с ключевыми словами, определяющими типы полей (CHARACTER, INTEGER, DATETIME и т.д.), предложением CONSTRAINT для создания ограничений на значения полей или связей между таблицами, ключевым словом UNIQUE, задающим свойство уникальности индекса таблицы, а также ключевого слова PRIMERYKEY, определяющего ключевое поле создаваемой таблицы.

Запросы на добавление полей или индексов реализуются SQL-инструкцией ALTERTABLE с использованием зарезервированных слов ADDCOLUMN (добавить поле) и ADDCONSTRAIN (добавить индекс). Этим же запросом с помощью зарезервированного слова DROPCOLUMN можно удалить поле из существующей таблицы. Как правило, запросы на добавление полей также используются для создания внешних ключей, задающих связи-отношения между таблицами. С этой целью используются зарезервированные слова FOREIGNKEY и REFERENCES.

Запросы на удаление таблицы или индекса реализуются SQL-инструкцией DROPTABLE с указанием имени удаляемой таблицы или индекса.

Запросы на создание индекса реализуются SQL-инструкцией CREATEINDEX с использованием зарезервированного слова UNIQUE запрета повторения значений в индексируемом поле и необязательного предложения WITH с параметрами DISALLOWNULL и IGNORENULL для запрета/разрешения нулевых (пустых) значений в индексируемом поле. Зарезервированное слово PRIMERY позволяет определить создаваемый индекс ключом таблицы (при этом создаваемый индекс по умолчанию является уникальным, т. е. повторения значений не допускаются).

Суть идеи *механизма событий*, правил и процедур заключается в том, что они после определения хранятся вместе с данными. Ядро СУБД при любом изменении состояния базы данных проверяет, не произошли ли ранее «поставленные на учет» события, и, если они произошли, обеспечивает проверку соответствующих правил и запуск соответствующих процедур обработки.

Функционирование базы данных осуществляется следующим образом:



1. В базе данных определяются так называемые события, связанные с изменениями данных — добавление новой записи в определенную таблицу, изменение, удаление записи.

2. Для каждого события в базе данных определяются правила (триггеры) по проверке определенных условий состояния данных.;

3. В зависимости от результатов проверки правил в базе данных запускаются на выполнение предварительно определенные процедуры.

Процедуры представляют собой последовательности команд по обработке данных, имеющие отдельное смысловое значение, и могут реализовываться на упрощенном макроязыке (последовательность команд запуска запросов или выполнения других действий, например по открытию-закрытию форм, таблиц и т. п.) или на языке 4GL, встроенном в СУБД.

В отличие от традиционного подхода, когда специальные прикладные программы постоянно «опрашивают» базу данных для обнаружения ситуаций, требующих обработки, «событийная» техника более экономична и естественна в технологическом плане.

Архитектуры вычислительных систем. Архитектура системы команд

Архитектура вычислительной машины (англ. computer architecture) – концептуальная структура вычислительной машины, определяющая проведение обработки информации и включающая методы преобразования информации в данные и принципы взаимодействия технических средств и программного обеспечения.

Архитектура системы – совокупность свойств системы, существенных для пользования.

Архитектурой компьютера называется его описание на некотором общем уровне, включающее описание пользовательских возможностей программирования, системы команд, системы адресации, организации памяти и т.д. Архитектура определяет принципы действия, информационные связи и взаимное соединение основных логических узлов компьютера: процессора, оперативного ЗУ, внешних ЗУ и периферийных устройств. Общность архитектуры разных компьютеров обеспечивает их совместимость с точки зрения пользователя.

Архитектура современного персонального компьютера подразумевает такую логическую организацию аппаратных компонент компьютера, при которой все устройства связываются друг с другом через магистраль, включающую в себя шины данных, адресов и управления.

Наиболее распространены следующие архитектурные решения.

Классическая архитектура (архитектура фон Неймана) — одно арифметико-логическое устройство (АЛУ), через которое проходит поток данных, и одно устройство управления (УУ), через которое проходит поток команд — программа. Это однопроцессорный компьютер. К этому типу архитектуры относится и архитектура персонального компьютера с общей шиной. Все функциональные блоки здесь связаны между собой общей шиной, называемой также системной магистралью.

Физически магистраль представляет собой многопроводную линию с гнездами для подключения электронных схем. Совокупность проводов магистрали разделяется на отдельные группы: шину адреса, шину данных и шину управления.

Периферийные устройства (принтер и др.) подключаются к аппаратуре компьютера через специальные контроллеры — устройства управления периферийными устройствами.

Контроллер — устройство, которое связывает периферийное оборудование или каналы связи с центральным процессором, освобождая процессор от непосредственного управления функционированием данного оборудования.

Архитектура с параллельными процессорами. Здесь несколько АЛУ работают под управлением одного УУ. Это означает, что множество данных может обрабатываться по одной программе — то есть по одному потоку команд. Высокое быстродействие такой архитектуры можно получить только на задачах, в которых одинаковые вычислительные операции выполняются одновременно на различных однотипных наборах данных.

Шинная (магистральная) архитектура ЭВМ. Наличие интеллектуальных контроллеров внешних устройств стало важной отличительной чертой машин третьего и четвертого поколений. Контроллер можно рассматривать как специализированный процессор, управляющий работой внешнего устройства. Такой процессор имеет собственную систему команд. Например, контроллер накопителя на гибких магнитных дисках (дисковод) умеет позиционировать головку на нужную дорожку диска, читать или записывать сектор, форматировать дорожку и т.п. Результаты выполнения каждой операции заносятся во внутренние регистры памяти контроллера и могут быть в дальнейшем прочитаны центральным процессором. Центральный процессор при необходимости произвести обмен выдает задание на его осуществление контроллеру. Дальнейший обмен информацией может протекать под руководством контроллера без участия центрального процессора. Последний получает возможность «заниматься своим делом», т.е. выполнять программу дальше (если по данной задаче до завершения обмена ничего сделать нельзя, то можно в это время решать другую).

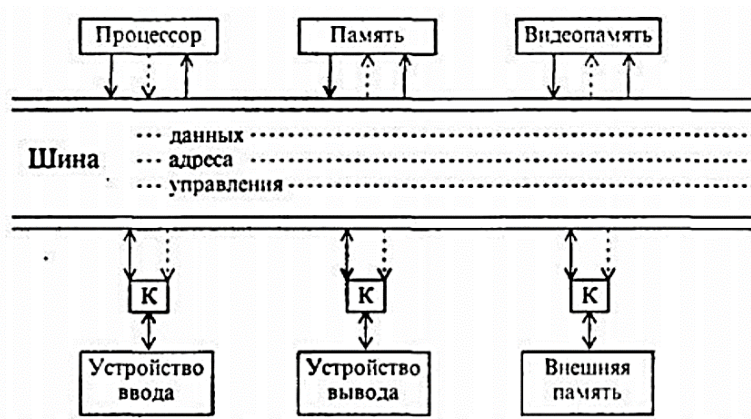


Рисунок 2 – Шинная (магистральная) архитектура ЭВМ

Из рисунка видно, что для связи между отдельными функциональными узлами ЭВМ используется общая шина (часто ее называют магистралью). Шина состоит из трех частей:

- шина данных, по которой передается информация;
- шина адреса, определяющая, куда передаются данные;
- шина управления, регулирующая процесс обмена информацией.

Отметим, что существуют модели компьютеров, у которых шины данных и адреса для экономии объединены. У таких машин сначала на шину выставляется адрес, а затем через некоторое время данные; для какой именно цели используется шина в данный момент, определяется сигналами на шине управления. Описанную схему легко пополнять новыми устройствами - это свойство называют открытостью архитектуры. Для пользователя открытая архитектура означает возможность свободно выбирать состав внешних устройств для своего компьютера, т.е. конфигурировать его в зависимости от круга решаемых задач.

Архитектура системы команд

Системой команд вычислительной машины называют полный перечень команд, которые способна выполнять данная ЭВМ. В свою очередь, под архитектурой системы команд (АСК) принято определять те средства вычислительной машины, которые видны и доступны программисту. АСК можно рассматривать как линию согласования нужд разработчиков программного обеспечения с возможностями создателей аппаратуры вычислительной машины. Таким образом, АСК служит интерфейсом между программной и аппаратной частями компьютера (см. рис. 5.6).

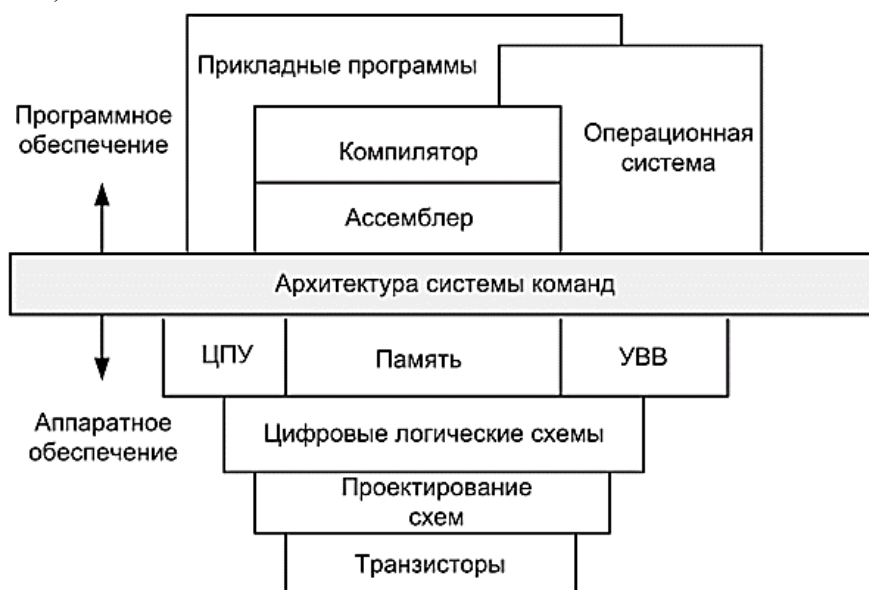


Рис 5.6. Основные компоненты компьютера

В конечном итоге цель разработчиков программного обеспечения и создателей аппаратуры — реализация вычислений наиболее эффективным образом, то есть за минимальное время, и здесь важнейшую роль играет правильный выбор архитектуры системы команд.

В упрощенной трактовке время выполнения программы $T_{\text{выч}}$ можно определить через число команд в программе $N_{\text{ком}}$, среднее количество тактов процессора CPI, приходящихся на одну команду и длительность тактового периода $\tau_{\text{тп}}$:

$$T_{\text{выч}} = N_{\text{ком}} * \text{CPI} * \tau_{\text{тп}}.$$

Каждая из составляющих данного выражения зависит от одних аспектов архитектуры системы команд и, в свою очередь, влияет на другие (см. рисунок 5.7), что свидетельствует о необходимости ответственного подхода к выбору АСК.

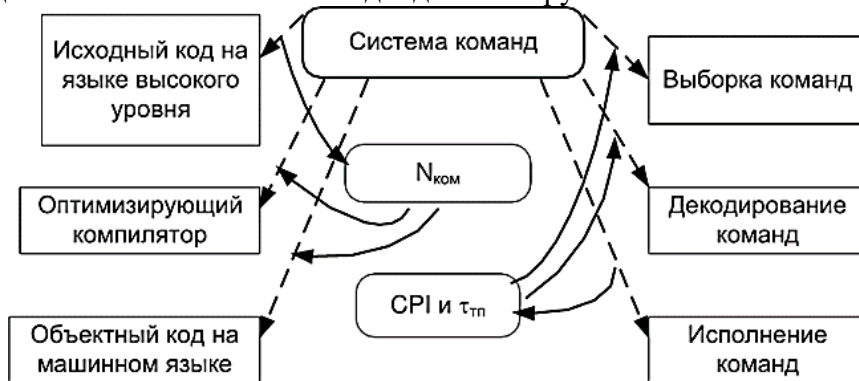


Рис 5.7. АСК и эффективность вычислений

В истории развития вычислительной техники отражаются изменения, происходившие во взглядах разработчиков на перспективность той или иной архитектуры системы команд. Сложившуюся на настоящий момент ситуацию в области АСК иллюстрирует рисунок 5.8.



Рис 5.8. Классификация АСК

Среди мотивов, чаще всего определяющих переход к новому типу АСК, остановимся на двух наиболее существенных. Первый — место хранения операндов, что влияет на количество и длину адресов, указываемых в адресной части команд обработки данных. Второй — это состав операций, выполняемых вычислительной машиной, и их сложность. Именно эти моменты взяты в качестве критериев для приведенной классификации архитектур системы команд. Важную роль при выборе АСК играет ответ на вопрос о том, где могут храниться операнды и каким образом к ним осуществляется доступ. С этих позиций различают следующие виды архитектур системы команд:

- стековую;
- аккумуляторную;
- регистровую;

- с выделенным доступом к памяти.

Выбор той или иной архитектуры влияет на принципиальные моменты: сколько адресов будет содержать адресная часть команд, какова будет длина этих адресов, насколько просто будет происходить доступ к операндам и какой будет общая длина команд.

Классификация современных вычислительных систем

Вычислительная система (ВС) - это взаимосвязанная совокупность аппаратных средств вычислительной техники и программного обеспечения, предназначенная для обработки информации.

Иногда под ВС понимают совокупность технических средств ЭВМ, в которую входит не менее двух процессоров, связанных общностью управления и использования общесистемных ресурсов (память, периферийные устройства, программное обеспечение и т.п.).

Ресурсы вычислительной системы

К ресурсам вычислительной системы относят такие средства вычислительной системы, которые могут быть выделены процессу обработки данных на определенный квант времени. Основными ресурсами ВС являются процессоры, области оперативной памяти, наборы данных, периферийные устройства, программы.

Виды вычислительных систем

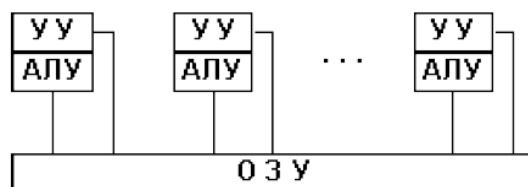
В зависимости от ряда признаков различают следующие вычислительные системы (ВС):

- однопрограммные и многопрограммные (в зависимости от количества программ, одновременно находящихся в оперативной памяти);
- индивидуального и коллективного пользования (в зависимости от числа пользователей, которые одновременно могут использовать ресурсы ВС);
- с пакетной обработкой и разделением времени (в зависимости от организации и обработки заданий);
- однопроцессорные, многопроцессорные и многомашинные (в зависимости от числа процессоров);
- сосредоточенные, распределенные (вычислительные сети) и ВС с теледоступом (в зависимости от территориального расположения и взаимодействия технических средств);
- работающие или не работающие в режиме реального времени (в зависимости от соотношения скоростей поступления задач в ВС и их решения);
- универсальные, специализированные и проблемно-ориентированные (в зависимости от назначения).

Способы организации и типы ВС

Вычислительные системы могут строиться на основе целых компьютеров или отдельных процессоров. В первом случае ВС будет многомашинной, во втором – многопроцессорной.

Многопроцессорная архитектура. Наличие в компьютере нескольких процессоров означает, что параллельно может быть организовано много потоков данных и много потоков команд. Таким образом, параллельно могут выполняться несколько фрагментов одной задачи.



Многомашинная вычислительная система. Здесь несколько процессоров, входящих в вычислительную систему, не имеют общей оперативной памяти, а имеют каждый свою (локальную). Каждый компьютер в многомашинной системе имеет классическую архитектуру, и такая система применяется достаточно широко. Однако эффект от применения такой вычислительной системы может быть получен только при решении задач, имеющих очень специальную структуру: она должна разбиваться на столько слабо связанных подзадач, сколько компьютеров в системе.

Преимущество в быстрой реакции многопроцессорных и многомашинных вычислительных

систем перед однопроцессорными очевидно.

Быстродействие и надежность многопроцессорных ВС по сравнению с многомашиными, взаимодействующими на уровне каналов связи, существенно повышаются, во-первых, ввиду ускоренного обмена информацией между процессорами, более быстрого реагирования на ситуации, возникающие в системе; во-вторых вследствие большей степени резервирования устройств системы (система сохраняет работоспособность, пока работоспособны хотя бы по одному модулю каждого типа устройств).

Типичным примером многопроцессорных ВС могут служить компьютерные сети, примером многопроцессорных вычислительных систем (МПВС) – суперкомпьютеры.

Параллельная обработка информации: уровни и способы организации

Общий метод увеличения производительности – организация параллельной обработки информации, т. е. одновременное решение задач или совмещение во времени этапов решения одной задачи.

Существуют следующие **уровни параллелизма**:

1) Микроуровневый параллелизм – основан на конвейеризации (увеличения числа инструкций, выполняемых в единицу времени путем создания конвейера команд) вычислений. Обеспечивается самим CPU.

2) Параллелизм уровня команд – реализуется посредством размещения в CPU сразу нескольких конвейеров. Обеспечивается самим CPU, и частично программистом. При этом для распараллеливания исполнения команд используются такие методики, как предсказатель переходов, обработка сразу 2-х веток условного оператора, развертка циклов и др.

3) Параллелизм уровня потоков и уровня заданий – применяется в процессорах класса MIMD. Параллелизм уровня потоков частично обеспечивается программистом частично ОС. Параллелизм уровня заданий обеспечивается ОС.

Способы организации. Во всем многообразии способов организации параллельной обработки можно выделить три основных направления:

- совмещение во времени различных этапов разных задач – это мультипрограммная обработка информации. Мультипрограммная обработка возможна даже в однопроцессорной ЭВМ и широко используется в современных СОД;

- одновременное решение различных задач или частей одной задачи – возможен только при наличии нескольких обрабатывающих устройств. При этом используются те или иные особенности задач или потоков задач, что позволяет осуществить тот или иной параллелизм;

- конвейерная обработка информации.

Можно выделить несколько типов параллелизма, отражающих эти особенности.

Типы параллелизма:

1. *Естественный параллелизм независимых задач* заключается в том, что в систему поступает непрерывный поток не связанных между собой задач, т. е. решение любой задачи не зависит от результатов решения других задач. В этом случае использование нескольких обрабатывающих устройств при любом способе комплексирования (косвенном или прямом) повышает производительность системы.

2. *Параллелизм независимых ветвей* – один из наиболее распространенных типов параллелизма в обработке информации. Суть его заключается в том, что при решении большой задачи могут быть выделены отдельные независимые части – ветви программы, которые при наличии нескольких обрабатывающих устройств могут выполняться параллельно и независимо друг от друга.

Двумя независимыми ветвями программы будем считать такие части задачи, при выполнении которых выполняются следующие условия:

- 1.) отсутствие функциональных связей;
- 2.) отсутствие связи по использованию одних и тех же полей оперативной памяти;
- 3.) независимость по управлению;
- 4.) программная независимость.

3. *Параллелизм объектов или данных* имеет место тогда, когда по одной и той же (или почти по одной и той же) программе должна обрабатываться некоторая совокупность данных, поступающих в систему одновременно. Это могут быть, например, задачи обработки сигналов от радиолокационной станции: все сигналы обрабатываются по одной и той же программе. Другой пример – обработка информации от датчиков, измеряющих одновременно один и тот же параметр и установленных на нескольких однотипных объектах. Программы обработки данных могут быть различного объема и сложности, начиная от очень простых, содержащих несколько операций, до больших программ в сотни и тысячи операций.

Реализация в многомашинных и многопроцессорных ВС

Различие понятий многомашинной и многопроцессорной ВС поясняет рис.6.1. Многомашинная ВС (ММС) содержит несколько ЭВМ, каждая из которых имеет свою ОП и работает под управлением своей операционной системы, а также средства обмена информацией между машинами. Реализация обмена информацией происходит, в конечном счете, путем взаимодействия операционных систем машин между собой. Это ухудшает динамические характеристики процессов межмашинного обмена данными. Применение многомашинных систем позволяет повысить надежность вычислительных комплексов. При отказе в одной машине обработку данных может продолжать другая машина комплекса. Однако можно заметить, что при этом оборудование комплекса недостаточно эффективно используется для этой цели. Достаточно в системе, изображенной на рис.6.1,а в каждой ЭВМ выйти из строя по одному устройству (даже разных типов), как вся система становится неработоспособной.

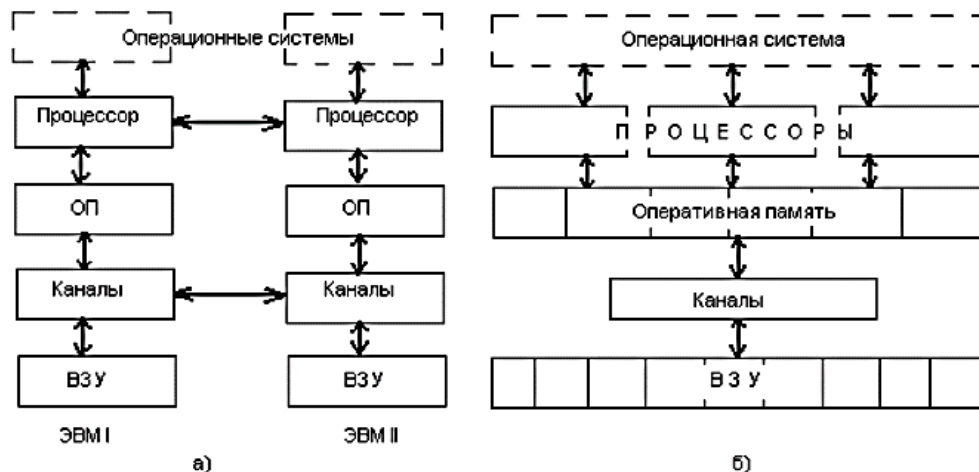


Рис. 6.1 Многомашинные(а) и многопроцессорные(б) системы.

Этих недостатков лишены многопроцессорные системы (МПС). В таких системах (рис. 6.1,б) процессоры обретают статус рядовых агрегатов вычислительной системы, которые подобно другим агрегатам, таким, как модули памяти, каналы, периферийные устройства, включаются в состав системы в нужном количестве.

Вычислительная система называется многопроцессорной, если она содержит несколько процессоров, работающих с общей ОП (общее поле оперативной памяти) и управляется одной общей операционной системой. Часто в МПС организуется общее поле внешней памяти.

Операционные контейнеры

При использовании контейнеров сначала на систему устанавливается ОС хоста, например Linux, а затем поверх нее устанавливается слой контейнеров — как правило, менеджер контейнеров, например **Docker**. Менеджер контейнеров, по сути, обеспечивает функцию гипервизора для контейнеров.

После установки слоя для контейнеров администраторы могут создавать экземпляры контейнеров из доступных вычислительных ресурсов системы и развертывать компоненты корпоративных приложений в контейнерах. Однако каждое контейнерное приложение использует одну и ту же базовую ОС: единую ОС хоста. Хотя слой контейнеров обеспечивает уровень

логической изоляции между контейнерами, общая ОС может представлять собой единую точку отказа для всех контейнеров в системе. Как и в случае с ВМ, контейнеры также легко переносятся между физическими системами при наличии подходящей ОС и среды контейнерного уровня.

Преимущества контейнеров

Контейнеры обладают своими уникальными свойствами и характеристиками:

Размер. Контейнеры используют одно общее ядро ОС и не используют собственные уникальные ОС, поэтому контейнеры являются гораздо меньшими логическими единицами, чем виртуальные машины. Это позволяет компьютеру одновременно размещать гораздо больше контейнеров, чем ВМ.

Скорость. Небольшой размер экземпляров контейнеров позволяет создавать и уничтожать их гораздо быстрее, чем ВМ. Благодаря этому контейнеры хорошо подходят для быстрого масштабирования и краткосрочного использования, что может быть нецелесообразно для ВМ.

Неизменяемость. В отличие от ВМ, контейнеры не изменяются. Вместо этого контейнерный оркестратор в программном слое контейнера запускает и останавливает контейнеры, когда они необходимы. Аналогичным образом, программное обеспечение, работающее в контейнерах, не обновляется, как традиционное ПО. Вместо этого обновления включаются в новый образ контейнера, который может быть развернут там, где это необходимо.

Недостатки контейнеров

Контейнеры обеспечили огромную масштабируемость и гибкость для корпоративных организаций, но есть и несколько недостатков:

Производительность. Контейнеров может быть много, и контейнеры используют общую ОС в дополнение к контейнерному слою. Это означает, что контейнеры эффективно используют ресурсы, но между контейнерами могут возникать разногласия при попытке получить доступ к аппаратным ресурсам, таким как сети. Такое соперничество может повлиять на общую производительность контейнера.

Совместимость. Контейнеры, упакованные для одной платформы, такие как Docker, могут не работать с другими платформами. Аналогично, некоторые контейнерные инструменты могут не работать с различными контейнерными платформами. Например, Red Hat OpenShift работает только с оркестратором Kubernetes. Учитывайте экосистему контейнеров при оценке контейнерных технологий для предприятия.

Хранение. Контейнеры изначально спроектированы так, чтобы быть без статических данных — данные в контейнере исчезают, когда исчезает контейнер. Существуют способы обеспечения постоянного хранения для контейнеров, например, Docker Data Volumes, но вопрос постоянного хранения контейнеров часто рассматривается как нечто отдельное.

Пригодность. Контейнеры, как правило, представляют собой небольшие и гибкие структуры, лучше всего подходящие для компонентов приложений или микросервисов. Полнофункциональные корпоративные приложения обычно плохо работают в контейнерах. Учитывайте пригодность при планировании развертывания контейнеров.

Векторные, матричные, ассоциативные системы

Вычислительные системы класса SIMD:

<<SIMD-системы были первыми вычислительными системами, состоящими из большого числа процессоров, SIMD-системы во многом похожи на классические фон-неймановские ВМ: в них также имеется одно устройство управления, обеспечивающее последовательное выполнение команд программы. Различие касается стадии выполнения, когда общая команда транслируется множеству процессоров/>>

- Векторные вычислительные системы

При большой размерности массивов последовательная обработка элементов матриц занимает слишком много времени, что и приводит к неэффективности универсальных ВС для рассматриваемого класса задач. Для обработки массивов требуются вычислительные средства, позволяющие с помощью единой команды производить действие сразу над всеми элементами массивов — *средства векторной обработки*.

В средствах векторной обработки под вектором понимается одномерный массив

однотипных данных

При размещении матрицы в памяти все ее элементы заносятся в ячейки с последовательными адресами, причем данные могут быть записаны строка за строкой или столбец за столбцом

Для повышения скорости обработки векторов все функциональные блоки векторных процессоров строятся по конвейерной схеме, причем так, чтобы каждая ступень любого из конвейеров справлялась со своей операцией за один такт.

- Матричные ВС

Назначение матричных вычислительных систем во многом схоже с назначением векторных ВС — обработка больших массивов данных.

Матричный процессор интегрирует множество идентичных функциональных блоков (ФБ), логически объединенных в матрицу и работающих в SIMD-стиле.

Векторный процессор имеет встроенные команды для обработки векторов данных, что позволяет эффективно загрузить конвейер из функциональных блоков.

Не столь существенно, как конструктивно реализована матрица процессорных элементов - на едином кристалле или на нескольких.

- Ассоциативные ВС

Однако, в отличие от матричных систем, обращение к данным производится не по адресам, где хранятся эти данные, а по отличительным признакам, содержащимся в самих данных

Ассоциативным процессором называют специализированный процессор, реализованный на базе ассоциативного запоминающего устройства (АЗУ), где, как известно, доступ к информации осуществляется не по адресу операнда, а по отличительным признакам, содержащимся в самом операнде.

Ассоциативная вычислительная система (АВС) представляет собой многопроцессорную ВС, объединяющую множество ассоциативных процессоров, процессор управления, процессор ввода/вывода и основную память.

Однородные системы и среды: RISC-архитектуры

Однородные вычислительные системы - вычислительная система, состоящая из большого числа одинаковых элементарных вычислительных машин, соединенных регулярной программно-управляемой коммуникационной сетью - регулярным каналом.

Однородные вычислительные среды (ОВС) предназначены для массовой обработки информации и представляют собой совокупность простейших одинаковых логических автоматов, регулярным образом соединенных между собой и программно настраиваемых на выполнение некоторой функции.

Сама аббревиатура RISC расшифровывается как Restricted (Reduced) Instruction Set Computer, что переводится как «компьютер с сокращенным набором команд». «Сокращенный набор команд» вовсе не означает, что количество инструкций меньше, чем число команд CISC-кристаллов. Разница состоит в том, что любая инструкция платформы RISC является простой и выполняется за один такт (по крайней мере, должна выполняться). При этом длина команды является фиксированной. Например, 32 бита. Также у RISC имеется гораздо больше регистров общего назначения. Плюс для этой архитектуры характерна конвейеризация. Именно ее использование (вкуче с упрощенными командами) позволяет эффективно наращивать тактовую частоту процессоров RISC. Другим распространенным признаком RISC является архитектура загрузки/хранения, где доступ к памяти осуществляется только с помощью определенных инструкций.

Развитие архитектур, ориентированных на языковые средства и среду программирования

Среды программирования (или как их еще называют, среды разработки) - это программы, в которых программисты пишут свои программы. Иными словами, среда программирования служит для разработки (написания) программ и обычно ориентируется на конкретный язык или несколько

языков программирования.

Алгоритмические языки – это знаковые системы, предназначенные для описания процессов решения задач с помощью компьютера. Программа, записанная на алгоритмическом языке, задает компьютеру определенную последовательность действий.

По назначению алгоритмические языки делятся на языки программирования и *языковые средства* автоматизированной системы. Языки программирования применяются, преимущественно, для создания автоматизированных систем.

Языковые средства автоматизированной системы являются инструментом в руках обслуживающего персонала автоматизированных систем, а также неспециализированных пользователей.

Языковые средства автоматизированной системы обеспечивают возможность общения информационных работников, субъектов управления с системой в процессе накопления, обработки и выдачи информации путем описания входных сообщений, запросов, а также операций и процедур, выполняемых над информацией в различных режимах и на различных этапах ее переработки. *Основные языковые средства* автоматизированной системы включают в себя:

- средства формализованного представления информации, поступающей на вход системы для ее ввода в банк данных, - входной язык;
- средства формализованного представления информации, хранимой в банке данных, - информационный язык;
- средства формализованного представления информационных задач, решаемых системой, - язык запросов (информационно-поисковый язык);
- средства формализованного представления корректирующей информации (язык коррекции), а также процедур ее обработки;
- средства формализованного представления и редактирования выходной информации системы, - язык описания и редактирования выходных форм.

Вспомогательные языковые средства автоматизированной системы включают в себя аппарат настройки технологических схем, таблиц контроля и трансляции информации, ведения внутрисистемных промежуточных файлов, ведения и поддержания в актуальном состоянии служебной информации для различных режимов обработки содержательной информации и средств ее защиты от несанкционированного доступа

Хорошая архитектура это прежде всего *выгодная* архитектура, делающая процесс разработки и сопровождения программы более простым и эффективным. Программу с хорошей архитектурой легче расширять и изменять, а также тестировать, отлаживать и понимать.

Преимущества, которые обычно соотносятся с понятием хорошая архитектура:

Масштабируемость (Scalability) возможность расширять систему и увеличивать ее производительность, за счет добавления новых модулей.

Ремонтопригодность (Maintainability) изменение одного модуля не требует изменения других модулей

Заменяемость модулей (Swappability) модуль легко заменить на другой

Возможность тестирования (Unit Testing) модуль можно отсоединить от всех остальных и протестировать / починить

Переиспользование (Reusability) модуль может быть переиспользован в других программах и другом окружении

Сопровождаемость (Maintenance) разбитую на модули программу легче понимать и сопровождать

Архитектура — это *организация системы, воплощенная в ее компонентах, их отношениях между собой и с окружением*. Таким образом, хорошая архитектура это, прежде всего, **модульная/блочная архитектура**.

Основы метрической теории ВС

Метрическая теория ВС занимается количественной оценкой показателей, которые характеризуют качество структурной и функциональной организации системы. В рамках метрической теории исследуется влияние организации системы и режимов ее функционирования

на производительность, надежность, стоимость и другие ее характеристики, а также решаются задачи обоснования выбора варианта структурной и функциональной организации системы и оптимальных параметров и внешних характеристик входящих в ее состав устройств ЭВМ, т.е. выполняется структурная и параметрическая оптимизация ВС.

Единицей измерения производительности компьютера является время: компьютер, выполняющий тот же объем работы за меньшее время является более быстрым. Время выполнения любой программы измеряется в секундах. Часто производительность измеряется как скорость появления некоторого числа событий в секунду, так что меньшее время подразумевает большую производительность.

Однако в зависимости от того, что мы считаем, время может быть определено различными способами. Наиболее простой способ определения времени называется астрономическим временем, временем ответа (response time), временем выполнения (execution time) или прошедшим временем (elapsed time). Это задержка выполнения задания, включающая буквально все: работу процессора, обращения к диску, обращения к памяти, ввод/вывод и накладные расходы операционной системы. Однако при работе в мультипрограммном режиме во время ожидания ввода/вывода для одной программы, процессор может выполнять другую программу, и система не обязательно будет минимизировать время выполнения данной конкретной программы.

Для измерения времени работы процессора на данной программе используется специальный параметр - время ЦП (CPU time), которое не включает время ожидания ввода/вывода или время выполнения другой программы. Очевидно, что время ответа, видимое пользователем, является полным временем выполнения программы, а не временем ЦП. Время ЦП может далее делиться на время, потраченное ЦП непосредственно на выполнение программы пользователя и называемое пользовательским временем ЦП, и время ЦП, затраченное операционной системой на выполнение заданий, затребованных программой, и называемое системным временем ЦП.

В ряде случаев системное время ЦП игнорируется из-за возможной неточности измерений, выполняемых самой операционной системой, а также из-за проблем, связанных со сравнением производительности машин с разными операционными системами. С другой стороны, системный код на некоторых машинах является пользовательским кодом на других и, кроме того, практически никакая программа не может работать без некоторой операционной системы. Поэтому при измерениях производительности процессора часто используется сумма пользовательского и системного времени ЦП.

В большинстве современных процессоров скорость протекания процессов взаимодействия внутренних функциональных устройств определяется не естественными задержками в этих устройствах, а задается единой системой синхросигналов, вырабатываемых некоторым генератором тактовых импульсов, как правило, работающим с постоянной скоростью. Дискретные временные события называются тактами синхронизации (clock ticks), просто тактами (ticks), периодами синхронизации (clock periods), циклами (cycles) или циклами синхронизации (clock cycles). Разработчики компьютеров обычно говорят о периоде синхронизации, который определяется либо своей длительностью (например, 10 наносекунд), либо частотой (например, 100 МГц). Длительность периода синхронизации есть величина, обратная к частоте синхронизации.

Таким образом, время ЦП для некоторой программы может быть выражено двумя способами: количеством тактов синхронизации для данной программы, умноженным на длительность такта синхронизации, либо количеством тактов синхронизации для данной программы, деленным на частоту синхронизации.

Важной характеристикой, часто публикуемой в отчетах по процессорам, является среднее количество тактов синхронизации на одну команду - CPI (clock cycles per instruction). При известном количестве выполняемых команд в программе этот параметр позволяет быстро оценить время ЦП для данной программы.

Таким образом, производительность ЦП зависит от трех параметров: такта (или частоты) синхронизации, среднего количества тактов на команду и количества выполняемых команд. Невозможно изменить ни один из указанных параметров изолированно от другого, поскольку базовые технологии, используемые для изменения каждого из этих параметров, взаимосвязаны: частота синхронизации определяется технологией аппаратных средств и функциональной организацией процессора; среднее количество тактов на команду зависит от функциональной

организации и архитектуры системы команд; а количество выполняемых в программе команд определяется архитектурой системы команд и технологией компиляторов. Когда сравниваются две машины, необходимо рассматривать все три компонента, чтобы понять относительную производительность.

Технология распределенной обработки данных

Распределенная обработка данных – обработка данных, выполняемая на независимых, но связанных между собой компьютерах, представляющих распределенную систему.

Для реализации распределенной обработки данных были созданы *многомашинные ассоциации*, структура которых разрабатывается по одному из следующих направлений:

- многомашинные вычислительные комплексы;
- компьютерные (вычислительные) сети.

Направления в технологиях распределенных систем:

- технология «клиент-сервер»;
- технологии объектного связывания;
- технологии реплицирования.

Технология «клиент-сервер». При реализации данной технологии отступают от одного из основных принципов создания распределенных систем – отсутствия центрального узла. Принцип централизации хранения и обработки данных является базовым принципом технологии клиент-сервер.

Один из основных принципов технологии «клиент-сервер» заключается в разделении операций обработки данных на три группы, имеющие различную природу. Первая группа – это ввод и отображение данных. Вторая группа объединяет прикладные операции обработки данных, характерные для решения задач данной предметной области. К третьей группе относятся операции хранения и управления данными (базами данных или файловыми системами).

Технологии объектного связывания. С узкой точки зрения, *технология объектного связывания данных* решает задачу обеспечения доступа из одной локальной базы, открытой одним пользователем, к данным в другой локальной базе (в другом файле), возможно находящейся на другой вычислительной установке, открытой и эксплуатируемой другим пользователем.

Решение этой задачи основывается на поддержке современными «настольными» СУБД (MS Access, MS FoxPro, dBase и др.) технологии «объектов доступа к данным» — DAO (Data Access Objects). Под объектом понимается интеграция данных и методов их обработки в одно целое (объект), на чем, как известно, основываются технологии объектно-ориентированного программирования. Другими словами, СУБД, поддерживающие DAO, получают возможность внедрять и оперировать в локальных базах объектами доступа к данным, физически находящимся в других файлах, возможно на других вычислительных установках и под управлением других СУБД. Связанные объекты для пользователя ничем не отличаются от внутренних объектов. Пользователь может также открывать связанные во внешних базах таблицы данных, осуществлять поиск, изменение, удаление и добавление данных, строить запросы по таким таблицам и т. д. Связанные объекты можно интегрировать в схему внутренней базы данных, т. е. устанавливать связи между внутренними и связанными таблицами.

Технологии реплицирования

Репликой называют особую копию базы данных для размещения на другом компьютере сети с целью автономной работы пользователей с одинаковыми (согласованными) данными общего пользования.

Основная идея реплицирования заключается в том, что пользователи работают автономно с одинаковыми (общими) данными, растажированными по локальным базам данных. Производительность работы системы повышается из-за отсутствия необходимости передачи и обмена данными по сети.

Программное обеспечение СУБД для реализации такого подхода соответственно дополняется функциями тиражирования (реплицирования) баз данных.

При этом, однако, возникают проблемы обеспечения одного из основополагающих принципов построения и функционирования распределённых систем, а именно – принципа

непрерывности согласованного состояния данных.

Технологии репликации данных в тех случаях, когда не требуется обеспечивать большие потоки и интенсивность обновляемых в информационной сети данных, являются экономичным решением проблемы создания распределенных информационных систем с элементами централизации по сравнению с использованием дорогостоящих клиент-серверных систем.

На практике для совместной коллективной обработки данных применяются смешанные технологии, включающие элементы объектного связывания данных, репликаций и клиент-серверных решений. Развитие и все более широкое распространение распределенных информационных систем является основной перспективой развития автоматизированных информационных систем.

Задачи сортировки Анализ сложности и эффективности алгоритмов сортировки

Задачи сортировки

Задачу сортировки следует понимать, как процесс перегруппировки однотипных элементов структуры данных в некотором определенном порядке. Цель сортировки - облегчить последующий поиск, обновление, исключение, включение элементов в структуру данных.

Разработано множество алгоритмов сортировки, однако нет алгоритма, который был бы наилучшим в любом случае.

Эффективность алгоритма сортировки может зависеть от ряда факторов, таких, как: число сортируемых элементов; диапазон и распределение значений сортируемых элементов; степень отсортированности элементов; характеристики алгоритма (сложность, требования к памяти и тп.); место размещения элементов (в оперативной памяти или на ВЗУ).

Сортируемые элементы часто представляют собой записи, данных определенной структуры. Каждая запись имеет поле ключа, по значению которого осуществляется сортировка, и поля данных. При рассмотрении алгоритмов сортировки нас интересует только поле ключа, поэтому другие поля опускаются из рассмотрения.

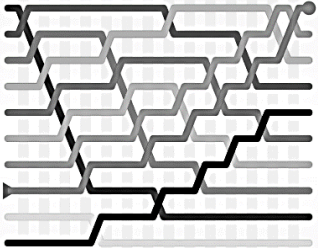
Метод сортировки называется устойчивым, если в процессе сортировки относительное расположение элементов с одинаковыми (равными) ключами не изменяется. Устойчивость сортировки желательна, если речь идет об элементах, уже отсортированных по некоторым другим ключам (свойствам), не влияющим на ключ, по которому сейчас осуществляется сортировка.

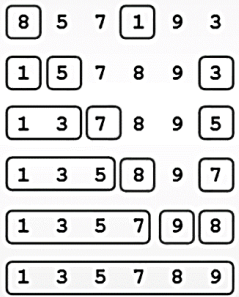

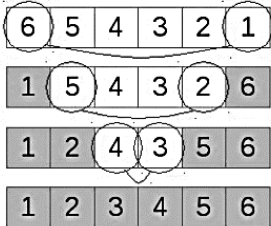
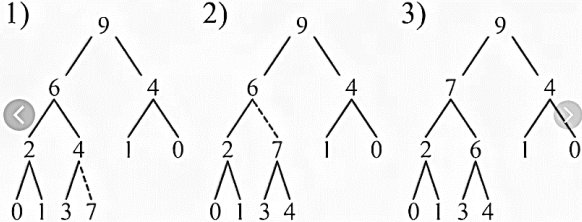
Внутренняя и внешняя сортировки

В зависимости от фактора размещения элементов все методы сортировки разбивают на два класса: внутренняя сортировка (сортировка массивов) и внешняя сортировка (сортировка файлов, или сортировка последовательностей).

Внешняя сортировка оперирует с запоминающими устройствами большого объема, но с доступом не произвольным, а последовательным, т. е. в данный момент мы «видим» только один элемент, а затраты на перемотку по сравнению с памятью неоправданно велики. Это накладывает некоторые дополнительные ограничения на алгоритм и приводит к специальным методам упорядочения, обычно использующим дополнительное дисковое пространство. Кроме того, доступ к данным на носителе производится намного медленнее, чем операции с оперативной памятью.

Алгоритмы сортировки

Сортировка пузырьком	<p>Один из простейших методов сортировки. Заключается в постепенном смещении элементов с большим значением в конец массива. Элементы последовательно сравниваются попарно, и если порядок в паре нарушен – меняются местами.</p> <p>Алгоритм проходит по заданному массиву множество раз, каждый раз сравнивая пару соседних друг с другом чисел. Если числа стоят не в том порядке, в котором должны, он меняет их местами.</p> <p>Элементы сортируемого массива «всплывают», как пузырьки воздуха в воде.</p> 
-----------------------------	---

Сортировка выбором	<p>Алгоритм ищет наименьший элемент в текущем списке и производит обмен его значения со значением первой неотсортированной позиции. То же самое происходит со вторым элементом с наименьшим значением. Цикл повторяется до тех пор, пока все элементы не займут нужную последовательность. Алгоритм ищет минимальный (или максимальный) элемент сортируемого массива и меняет его местами с первым несортированным элементом. Стандартный поиск минимума для каждого из n элементов – это квадрат. Сложность: $O(n^2)$</p> <p>Selection Sort.</p> 
Быстрая сортировка	<p>Считается одним из самых быстрых алгоритмов сортировки. Как и сортировка слиянием, работает по принципу «разделяй и властвуй». Временная сложность алгоритма может достигать $O(n \log n)$.</p> <p>Ещё один рекурсивный алгоритм. Сначала проводится грубая оценка массива на основе некоторого элемента, называемого опорным. Все элементы, если сортируем по возрастанию, большие опорного, перекидываются направо от него, а все меньшие – налево. Массив делится на две части, каждая из которых сортируется отдельно.</p> <p> Ч.Э.Хоар</p> <p>Идея: выгоднее переставлять элементы, который находятся дальше друг от друга.</p> 
Сортировка кучей (Пирамидальная сортировка)	<p>Имея построенную пирамиду, несложно реализовать сортировку. Так как корневой узел пирамиды имеет самое большое значение, мы можем отделить его и поместить в конец массива. Вместо корневого узла можно поставить последний узел дерева и, просеив его вниз, снова получить пирамиду. В новом дереве корень имеет самое большое значение среди оставшихся элементов. Его снова можно отделить, и так далее. Алгоритм получается следующий:</p> <ol style="list-style-type: none"> 1. поменять местами значения первого и последнего узлов пирамиды; 2. отделить последний узел от дерева, уменьшив размер дерева на единицу (элемент остаётся в массиве); 3. восстановить пирамиду, просеив вниз её новый корневой элемент; 4. перейти к пункту 1; <p>По мере работы алгоритма, часть массива, занятая деревом, уменьшается, а в конце массива накапливается отсортированный результат.</p> <p>Процесс просеивания:</p> 

<p>Сортировка вставками</p>	<p>Применяется для вставки элементов массива на «свое место». Сортировка вставками представляет собой простой метод сортировки и используется для раскладки колоды во время игры в бридж.</p> <p>Каждый элемент массива вставляется на правильное (относительно остальных) место в новом, отсортированном массиве.</p> <p>Так как кроме обхода всех элементов массива, для каждого элемента нужно найти место в новом массиве и сдвинуть некоторые элементы (после вставки, но до старого расположения элемента), сложность задачи – $O(n^2)$</p>
<p>Сортировка слиянием</p>	<p>Следует принципу «разделяй и властвуй», согласно которому массив данных разделяется на равные части, которые сортируются по-отдельности. После они сливаются, в результате получается отсортированный массив.</p> <p>Сортируемый массив разбивается на две части, каждая из которых сортируется отдельно (возможно, этим же самым алгоритмом), а потом результаты сортировок объединяются в один. Это – рекурсивный алгоритм.</p> <p>Главный принцип этого алгоритма: массив из одного элемента уже отсортирован.</p>

Анализ сложности и эффективности алгоритмов поиска и сортировки

Критериями оценки эффективности алгоритма сортировки является пространственная и временная сложность.

Пространственная сложность

Означает количество памяти, затраченной на выполнение алгоритма. Пространственная сложность включает вспомогательную память и память для хранения входных данных.

Временная сложность

Означает время, за которое алгоритм выполняет поставленную задачу с учетом входных данных.

В таблице представлена оценка сложности алгоритмов

Алгоритм сортировки	Время работы в худшем случае	Время работы в среднем случае	Время работы в лучшем случае	Пространственная сложность
Сортировка пузырьком	n^2	n^2	n	1
Сортировка выбором	n^2	n^2	n^2	1
Быстрая сортировка	n^2	$n \cdot \log n$	$n \cdot \log n$	$n \cdot \log n$
Сортировка кучей	$n \cdot \log n$	$n \cdot \log n$	$n \cdot \log n$	1
Сортировка вставками	n^2	n^2	n	1
Сортировка слиянием	$n \cdot \log n$	$n \cdot \log n$	$n \cdot \log n$	n

У каждого алгоритма сортировки своя временная и пространственная сложность. Использовать можно любой из представленных алгоритмов в зависимости от поставленных задач. Но по моему субъективному мнению лучшим алгоритмом является быстрая сортировка. Она позволяет выбрать опорный элемент и разделяет массив на 3 части: меньше, равно и больше опорного элемента.

Современные технологии разработки программного обеспечения Управление версиями Документирование

Современные технологии разработки программного обеспечения, постановка задачи, оценка осуществимости

Технология разработки программного обеспечения – это совокупность процессов и методов создания программного продукта в заданные сроки и с заданными характеристиками качества.

Данная деятельность включает в себя несколько этапов, с которыми так или иначе придётся столкнуться при разработке достаточно крупного ПО.

Любой *этап жизненного цикла* имеет четко определенные критерии начала и окончания. Состав этапов *жизненного цикла*, а также критерии, в конечном итоге определяющие последовательность этапов *жизненного цикла*, определяется коллективом разработчиков и/или заказчиком. В настоящее время существует несколько основных моделей *жизненного цикла*, которые могут быть адаптированы под реальную разработку.

1.3.1. Каскадный жизненный цикл

Каскадный жизненный цикл (иногда называемый водопадным) основан на постепенном увеличении степени детализации описания всей разрабатываемой системы. Каждое повышение степени детализации определяет переход к следующему состоянию разработки (Рис. 1.1).



Рис. 1.1. Каскадная модель жизненного цикла

На первом этапе составляется концептуальная структура системы, описываются общие принципы ее построения, правила взаимодействия с окружающим миром, - определяются системные требования.

На втором этапе по системным требованиям составляются требования к программному обеспечению - здесь основное внимание уделяется функциональности программной компоненты, программным интерфейсам. Естественно, все программные комплексы выполняются на какой-либо аппаратной платформе. Если в ходе проекта требуется также разработка аппаратной компоненты, параллельно с требованиями к программному обеспечению идет подготовка требований к аппаратному обеспечению.

На третьем этапе на основе требований к программному обеспечению составляется детальная спецификация архитектуры системы - описываются разбиение системы по конкретным модулям, интерфейсы между ними, заголовки отдельных функций и т.п.

На четвертом этапе пишется программный код, соответствующий детальной спецификации, на пятом этапе выполняется тестирование - проверка соответствия программного кода требованиям, определенным на предыдущих этапах.

Особенность каскадного жизненного цикла состоит в том, что переход к следующему этапу происходит только тогда, когда полностью завершены все работы предыдущего этапа. То есть сначала полностью готовятся все требования к системе, затем по ним полностью готовятся все требования к программному обеспечению, полностью разрабатывается архитектура системы и так далее до тестирования.

Естественно, что в случае достаточно больших систем такой подход себя не оправдывает. Работа на каждом этапе занимает значительное время, а внесение изменений в первичные документы либо невозможно, либо вызывает лавинообразные изменения на всех других этапах.

Как правило, используется модификация каскадной модели, допускающая возврат на любой из ранее выполненных этапов. При этом фактически возникает дополнительная процедура

принятия решения.

Действительно если тесты обнаружили несоответствие реализации требованиям, то причина может крыться: (а) в неправильном тесте, (б) в ошибке кодирования (реализации), (в) в неверной архитектуре системы, (г) в некорректности требований к программному обеспечению и т.д. Все эти случаи требуют анализа, для того чтобы принять решение о том, на какой этап жизненного цикла надо возвратиться для устранения обнаруженного несоответствия.

1.3.2. V-образный жизненный цикл

В качестве своеобразной "работы над ошибками" классической каскадной модели стала применяться модель жизненного цикла, содержащая процессы двух видов - основные процессы разработки, аналогичные процессам каскадной модели, и процессы верификации, представляющие собой цепь обратной связи по отношению к основным процессам (Рис. 1.2).

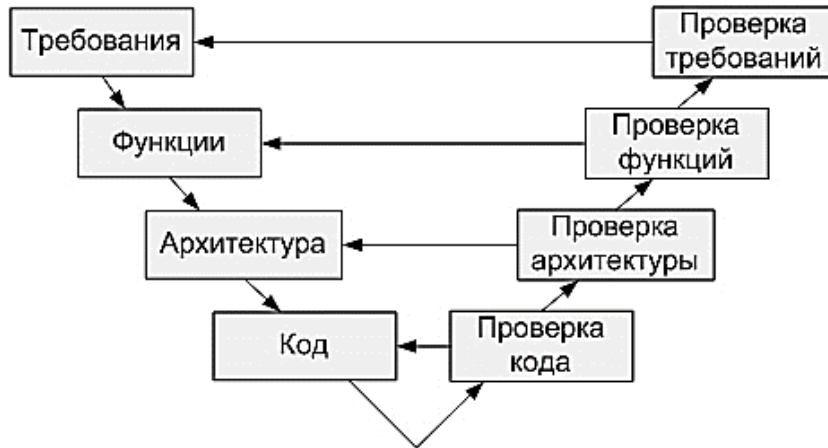


Рис. 1.2. V-образный жизненный цикл

Таким образом, в конце каждого этапа жизненного цикла разработки, а зачастую и в процессе выполнения этапа, осуществляется проверка взаимной корректности требований различных уровней. Данная модель позволяет более оперативно проверять корректность разработки, однако, как и в каскадной модели, предполагается, что на каждом этапе разрабатываются документы, описывающие поведение всей системы в целом.

1.3.3. Спиральный жизненный цикл

Оба рассмотренных типа жизненных циклов предполагают, что заранее известны все требования пользователей или, по крайней мере, предполагаемые пользователи системы настолько квалифицированы, что могут высказывать свои требования к будущей системе, не видя ее перед глазами.

Естественно, такая картина достаточно утопична, поэтому постепенно появилось решение, исправляющее основной недостаток V-образного жизненного цикла - предположение о том, что на каждом этапе разрабатывается очередное полное описание системы. Этим решением стала *спиральная модель жизненного цикла* (Рис. 1.3).



Рис. 1.3. Спиральный жизненный цикл

В спиральной модели разработка системы происходит повторяющимися этапами - витками

спирали. Каждый виток спирали - один каскадный или V-образный жизненный цикл. В конце каждого витка получается законченная версия системы, реализующая некоторый набор функций. Затем она предъявляется пользователю, на следующий виток переносится вся документация, разработанная на предыдущем витке, и процесс повторяется.

Таким образом, система разрабатывается постепенно, проходя постоянные согласования с заказчиком. На каждом витке спирали функциональность системы расширяется, постепенно дорастая до полной.

1.3.4. Экстремальное программирование

Реалии последних лет показали, что для систем, требования к которым изменяются достаточно часто, необходимо еще больше уменьшить длительность витка спирального жизненного цикла. В связи с этим сейчас стали весьма популярными быстрые жизненные циклы разработки, например, жизненный цикл в методологии *eXtreme Programming* (XP).

Основная идея жизненного цикла экстремального подхода - максимальное укорачивание длительности одного этапа жизненного цикла и тесное взаимодействие с заказчиком. По сути, на каждом этапе происходит реализация и тестирование одной функции системы, после завершения которых система сразу передается заказчику на проверку или эксплуатацию.

Основная проблема данного подхода - интерфейсы между модулями, реализующими эту функцию. Если во всех предыдущих типах жизненного цикла интерфейсы достаточно четко определяются в самом начале разработки, поскольку заранее известны все модули, то при экстремальном подходе интерфейсы проектируются "на лету", вместе с разрабатываемыми модулями.

При формулировании цели и задач проекта должен быть создан документ, дающий ответы на следующие вопросы:

- Как определяется предметная область для данного проекта? Какие термины используются в предметной области? Какие бизнес-процессы, затрагиваемые проектом, протекают в предметной области?
- Какая цель у проекта? Какой эффект должна оказать разрабатываемая система на бизнес-процессы предметной области?
- Какие задачи требуется решить, чтобы достичь поставленной цели? По каким критериям будет оцениваться качество решения поставленных задач? Каковы функциональные и нефункциональные показатели качества разрабатываемой системы?
- Какие ограничения на сроки выполнения, ресурсы, бюджет накладываются на реализацию проекта?

Планирование, тестирование, обеспечение оценки качества

Планирование является целью каждого вида деятельности, когда мы хотим обнаружить то, что относится к проекту. Важной задачей при создании программного обеспечения является извлечение требований или анализ требований. Клиенты обычно имеют абстрактное представление о том, чего они хотят в конечном результате, но не знают, что должно делать *программное обеспечение*.

Тестирование – это процесс, содержащий в себе все активности жизненного цикла, как динамические, так и статические, касающиеся планирования, подготовки и оценки программного продукта и связанных с этим результатов работ, с целью определить, что они соответствуют описанным требованиям, показать, что они подходят для заявленных целей и для определения дефектов. Качество же – это способность ПО при заданных условиях удовлетворять установленным или предполагаемым потребностям.

Верификация - это процесс определения, выполняют ли *программные средства* и их компоненты требования, наложенные на них в последовательных этапах жизненного цикла разрабатываемой программной системы.

Основная цель верификации состоит в подтверждении того, что *программное обеспечение* соответствует требованиям. Дополнительной целью является выявление и *регистрация* дефектов и ошибок, которые внесены во время разработки или модификации программы.

Понятие качества ПС.

Качество ПС - это совокупность его черт и характеристик, которые влияют на его

способность удовлетворять заданные потребности пользователей.

Качество ПС является удовлетворительным, когда оно обладает указанными свойствами в такой степени, чтобы гарантировать успешное его использование.

Критерии качества ПС:

- функциональность,
- надежность,
- легкость применения,
- эффективность,
- сопровождаемость,
- мобильность.

Функциональность - это способность ПС выполнять набор функций, удовлетворяющих заданным или подразумеваемым потребностям пользователей. Набор указанных функций определяется во внешнем описании ПС.

Легкость применения - это характеристики ПС, которые позволяют минимизировать усилия пользователя по подготовке исходных данных, применению ПС и оценке полученных результатов, а также вызывать положительные эмоции определенного или подразумеваемого пользователя.

Эффективность - это отношение уровня услуг, предоставляемых ПС пользователю при заданных условиях, к объему используемых ресурсов.

Сопровождаемость - это характеристики ПС, которые позволяют минимизировать усилия по внесению изменений для устранения в нем ошибок и по его модификации в соответствии с изменяющимися потребностями пользователей.

Мобильность - это способность ПС быть перенесенным из одной среды (окружения) в другую, в частности, с одной ЭВМ на другую.

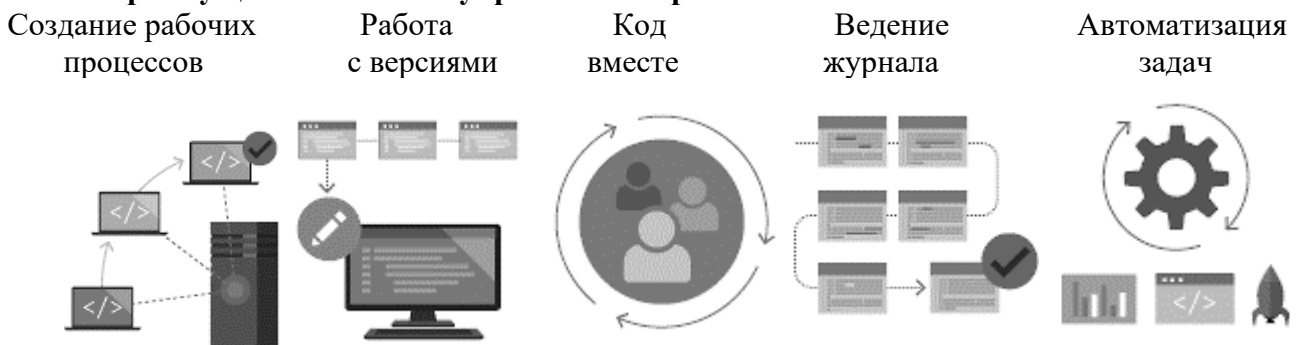
Обязательные критерии качества: функциональность и надежность.

Групповая разработка, управление версиями, организация коллектива разработчиков, документирование

Системы управления версиями — это программное обеспечение, помогающее отслеживанию изменений, внесенных в код с течением времени. Когда разработчик редактирует код, система управления версиями создает моментальный снимок файлов. Затем этот моментальный снимок сохраняется, чтобы при необходимости им было можно воспользоваться позже.

Без контроля версий разработчики могут удерживать на своем компьютере несколько копий кода. Это опасно, так как можно легко изменить или удалить файл в неправильной копии кода, что может привести к потере работы. Системы управления версиями решают эту проблему, управляя всеми версиями кода, но выполнив рабочую группу с одной версией за раз.

Преимущества системы управления версиями



Рабочие процессы управления версиями предотвращают Chaos всех пользователей, использующих собственный процесс разработки с различными и несовместимыми средствами. Системы управления версиями обеспечивают принудительную обработку и разрешения, чтобы все пользователи оставались на одной странице.

Каждая версия имеет описание того, что делают изменения в версии, например исправление ошибки или добавление функции. Эти описания помогают команде отслеживать изменения в коде по версии, а не по отдельным изменениям файла. Код, хранящийся в версиях, можно в любой

момент при необходимости просмотреть и восстановить из системы управления версиями. Это позволяет легко создать новую работу с любой версией кода.

Управление версиями синхронизирует версии и гарантирует, что изменения не конфликтуют с другими изменениями других. Группа использует систему управления версиями для устранения конфликтов, даже если пользователи делают изменения одновременно.

Управление версиями сохраняет историю изменений по мере того, как команда сохраняет новые версии кода. Этот журнал позволяет узнать, кто, зачем и когда внес изменения. Журнал дает группам уверенность в экспериментах, так как в любое время можно легко выполнить откат до предыдущей хорошей версии. Журнал позволяет любому пользователю выполнять базовые операции из любой версии кода, например для исправления ошибки в предыдущем выпуске.

Функции автоматизации системы управления версиями сохраняют время и создают противоречивые результаты. Автоматизируйте тестирование, анализ кода и развертывание, сохраняя новые версии в системе управления версиями.

Когда проектная команда включает более двух человек неизбежно встает вопрос о распределении ролей, прав и ответственности в команде. Конкретный набор ролей определяется многими факторами - количеством участников разработки и их личными предпочтениями, принятой методологией разработки, особенностями проекта и другими факторами. Практически в любом коллективе разработчиков можно выделить перечисленные ниже роли. Некоторые из них могут вовсе отсутствовать, при этом отдельные люди могут выполнять сразу несколько ролей, однако общий состав меняется мало.

Заказчик (заявитель). Эта роль принадлежит представителю организации, заказавшей разрабатываемую систему. Обычно заявитель ограничен в своем взаимодействии и общается только с менеджерами проекта и специалистом по сертификации или внедрению. Обычно заказчик имеет право изменять требования к продукту (только во взаимодействии с менеджерами), читать проектную и сертификационную документацию, затрагивающую нетехнические особенности разрабатываемой системы.

Менеджер проекта. Эта роль обеспечивает коммуникационный канал между заказчиком и проектной группой. Менеджер продукта управляет ожиданиями заказчика, разрабатывает и поддерживает бизнес-контекст проекта. Его работа не связана напрямую с продажей, он сфокусирован на продукте, его задача - определить и обеспечить *требования заказчика*. Менеджер проекта имеет право изменять требования к продукту и финальную документацию на продукт.

Менеджер программы. Эта роль управляет коммуникациями и взаимоотношениями в проектной группе, является в некотором роде координатором, разрабатывает функциональные спецификации и управляет ими, ведет *график* проекта и отчитывается по состоянию проекта, инициирует принятие критичных для хода проекта решений.

Менеджер программы имеет право изменять функциональные спецификации верхнего уровня, план-график проекта, *распределение ресурсов* по задачам. Часто на практике роль менеджера проекта и менеджера программы выполняет один человек.

Разработчик. Разработчик принимает технические решения, которые могут быть реализованы и использованы, создает продукт, удовлетворяющий спецификациям и ожиданиям заказчика, консультирует другие роли в ходе проекта. Он участвует в обзорах, реализует возможности продукта, участвует в создании функциональных спецификаций, отслеживает и исправляет ошибки за приемлемое время. В контексте конкретного проекта *роль разработчика* может подразумевать, например, установку программного обеспечения, настройку продукта или услуги. Разработчик имеет *доступ* ко всей проектной документации, включая документацию по тестированию, имеет право на изменение программного кода системы в рамках своих служебных обязанностей.

Специалист по тестированию. Специалист по тестированию определяет стратегию тестирования, тест-требования и тест-планы для каждой из фаз проекта, выполняет тестирование системы, собирает и анализирует отчеты о прохождении тестирования. Тест-требования должны покрывать *системные требования*, функциональные спецификации, требования к надежности и нагрузочной способности, пользовательские интерфейсы и собственно программный код. В реальности роль специалиста по тестированию часто разбивается на две - разработчика тестов и

тестировщика. Тестировщик выполняет все работы *по* выполнению тестов и сбору информации, разработчик тестов - всю остальные работы.

Специалист по контролю качества. Эта роль принадлежит члену проектной группы, который осуществляет взаимодействие с разработчиком, менеджером программы и специалистами *по* безопасности и сертификации с целью отслеживания целостной картины качества продукта, его соответствия стандартам и спецификациям, предусмотренным проектной документацией. Следует различать специалиста *по* тестированию и специалиста *по* контролю качества. Последний не является членом технического персонала проекта, ответственным за детали и технику работы. *Контроль* качества подразумевает в первую очередь *контроль* самих процессов разработки и проверку их соответствия определенным в стандартах качества критериям.

Специалист по сертификации. При разработке систем, к надежности которых предъявляются повышенные требования, перед вводом системы в эксплуатацию требуется подтверждение со стороны уполномоченного органа (обычно государственного) соответствия ее эксплуатационных характеристик заданным критериям. Такое соответствие определяется в ходе сертификации системы. Специалист *по* сертификации может либо быть представителем сертифицирующих органов, включенным в состав коллектива разработчиков, либо наоборот - представлять интересы разработчиков в сертифицирующем органе. Специалист *по* сертификации приводит документацию на программную систему в соответствие требованиям сертифицирующего органа либо участвует в процессе создания документации с учетом этих требований. Также специалист *по* сертификации ответственен за все взаимодействие между коллективом разработчиков и сертифицирующим органом. Важной особенностью роли является независимость специалиста от проектной группы на всех этапах создания продукта. Взаимодействие специалиста с членами проектной группы ограничивается менеджерами *по* проекту и *по* программе.

Специалист по внедрению и сопровождению. Участвует в анализе особенностей площадки заказчика, на которой планируется проводить внедрение разрабатываемой системы, выполняет весь спектр работ *по* установке и настройке системы, проводит обучение пользователей.

Специалист по безопасности. Данный специалист ответственен за весь спектр вопросов безопасности создаваемого продукта. Его работа начинается с участия в написании требований к продукту и заканчивается финальной стадией сертификации продукта.

Инструктор. Эта роль отвечает за снижение затрат на дальнейшее сопровождение продукта, обеспечение максимальной эффективности работы пользователя. Важно, что речь идет о производительности пользователя, а не системы. Для обеспечения оптимальной продуктивности инструктор собирает статистику *по* производительности пользователей и создает решения для повышения производительности, в том числе с использованием различных аудиовизуальных средств. Инструктор принимает участие во всех обсуждениях пользовательского интерфейса и архитектуры продукта.

Технический писатель. Лицо, осуществляющее эту роль, несет обязанности *по* подготовке документации к разработанному продукту, финального описания функциональных возможностей. Также он участвует в написании сопроводительных документов (системы помощи, руководство пользователя).

Документация, создаваемая на различных этапах жизненного цикла

Синхронизация всех этапов разработки происходит при помощи документов, которые создаются на каждом из этапов. Документация при этом создается и на прямом отрезке жизненного цикла - при разработке программной системы, и на обратном - при ее верификации. Попробуем на примере V-образного жизненного цикла проследить, какие типы документов создаются на каждом из отрезков и какие взаимосвязи между ними существуют (Рис 1.8).

Результатом этапа разработки требований к системе являются сформулированные требования к системе: документы, описывающие общие принципы работы системы, ее взаимодействие с "окружающей средой" - пользователями системы, а также, программными и аппаратными средствами, обеспечивающими ее работу. Обычно параллельно с требованиями к системе создается план верификации и определяется стратегия верификации. Эти документы определяют общий подход к тому, как будет выполняться тестирование, какие методики будут применяться, какие аспекты будущей системы должны быть подвергнуты тщательной проверке. Еще одна задача,

решаемая при помощи определения стратегии верификации - *определение* места различных верификационных процессов и их связей с процессами разработки.

Верификационный процесс, работающий с системными требованиями - это процесс валидации требований, сопоставления их реальным ожиданиям заказчика. Забегая вперед, скажем, что процесс валидации отличается от приемо-сдаточных испытаний, выполняемых при передаче готовой системы заказчику, хотя может считаться частью таких испытаний. *Валидация* является средством доказать не только *корректность* реализации системы с точки зрения заказчика, но и *корректность* принципов, положенных в основу ее разработки.



Рис. 1.8. Процессы и документы при разработке программных систем

Требования к системе являются основой для процесса разработки функциональных требований и архитектуры проекта. В ходе этого процесса разрабатываются общие требования к программному обеспечению системы, к функциям, которые она должна выполнять. *Функциональные требования* часто включают в себя *определение* моделей поведения системы в штатных и нештатных ситуациях, правила обработки данных и определения интерфейса пользователя. Текст требования, как правило, включает в себя слова "должна, должен" и имеет структуру вида "В случае, если *значение* температуры на датчике *ABC* достигает 30 и выше градусов Цельсия, система должна прекращать выдачу звукового сигнала". *Функциональные требования* являются основой для разработки архитектуры системы - описания ее структуры в терминах подсистем и структурных единиц языка, на котором производится реализация - областей, классов, модулей, функций и т.п.

На базе функциональных требований пишутся тест-требования - документы, содержащие *определение* ключевых точек, которые должны быть проверены для того, чтобы убедиться в корректности реализации функциональных требований. Часто тест-требования начинаются словами "Проверить, что" и содержат ссылки на соответствующие им *функциональные требования*. Примером тест-требований для приведенного выше функционального требования могут служить "Проверить, что в случае падения температуры на датчике *ABC* ниже 30 градусов Цельсия система выдает предупреждающий звуковой сигнал" и "Проверить, что в случае, когда *значение* температуры на датчике *ABC* выше 30 градусов Цельсия, система не выдает звуковой сигнал".

Одна из проблем, возникающая при написании тест-требований - принципиальная нетестируемость некоторых требований: например, требование "*Интерфейс* пользователя должен быть интуитивно понятным" невозможно проверить без четкого определения того, что является

интуитивно понятным интерфейсом. Такие неконкретные *функциональные требования* обычно впоследствии видоизменяют.

Архитектурные особенности системы также могут служить источником для создания тест-требований, учитывающих особенности программной реализации системы. Примером такого требования является, например, "Проверить, что *значение* температуры на датчике *ABC* не выходит за 255".

На основе функциональных требований и архитектуры пишется программный код системы, для его проверки на основе тест-требований готовится тест-план - описание последовательности тестовых примеров, выполняющих проверку *соответствия реализации* системы требованиям. Каждый тестовый пример содержит конкретное описание значений, подаваемых на вход системы, значений, которые ожидаются на выходе, и описание сценария выполнения теста.

В зависимости от объекта тестирования тест-план может быть подготовлен либо в виде программы на каком-либо языке программирования, либо в виде входного файла данных для инструментария, который выполняет тестируемую систему и передает ей значения, указанные в тест-плане, либо в виде инструкций для пользователя системы, описывающей необходимые действия, которые нужно выполнить для проверки различных функций системы.

В результате выполнения всех тестовых примеров собирается *статистика* об успешности прохождения тестирования - *процент* тестовых примеров, для которых реальные выходные значения совпали с ожидаемыми, так называемых пройденных тестов. Не пройденные тесты являются исходными данными для анализа причин ошибок и последующего их исправления.

На этапе интеграции осуществляется *сборка* отдельных модулей системы в единое целое и выполнение тестовых примеров, проверяющих всю функциональность системы.

На последнем этапе осуществляется поставка готовой системы заказчику. Перед внедрением специалисты заказчика совместно с разработчиками проводят приемо-сдаточные испытания - выполняют проверку критичных для пользователя функций согласно заранее утвержденной программе испытаний. При успешном прохождении испытаний система передается заказчику, в противном случае отправляется на доработку.

Структурное проектирование, CASE-средства, реинжиниринг программных систем

Структурное программирование – это процесс пошагового разбиения алгоритма на все более мелкие части, с целью получить такие элементы, для которых можно легко написать конкретные предписания.

Два принципа структурного программирования:

1. последовательная детализация «сверху – вниз»
2. ограниченность базового набора структур для построения алгоритмов любой степени сложности

Требования структурного программирования:

1. программа должна составляться мелкими шагами, таким образом, сложная задача разбивается на достаточно простые, легко воспринимаемые части.
2. логика программы должна опираться на минимальное число достаточно базовых управляющих структур (линейные, разветвляющиеся и циклические структуры)

Основные свойства и достоинства структурного программирования:

1. уменьшение сложности программ
2. возможность демонстрации правильности программ на различных этапах решения задачи
3. наглядность программ
4. простота модификации (внесения изменения) программ.

Современные средства программирования должны обеспечивать максимальную защиту от возможных ошибок разработчика.

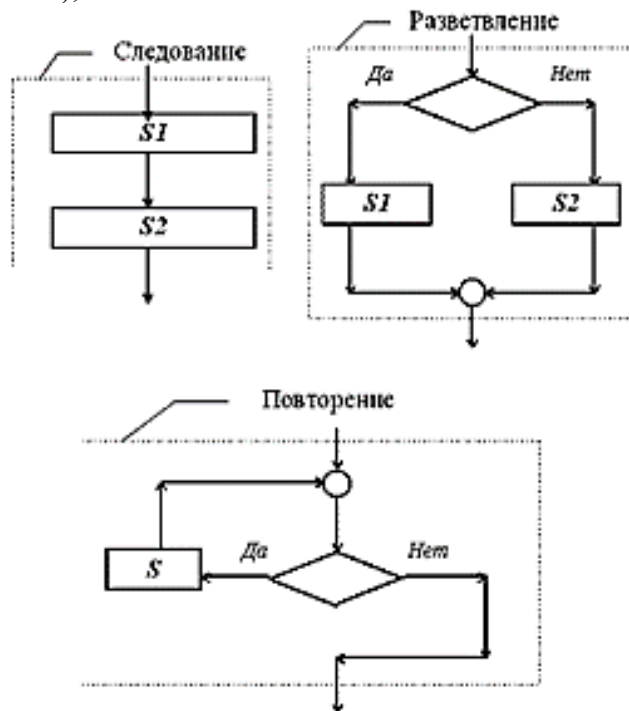
Тут можно провести аналогию с развитием методов управления автотранспортом. Сначала безопасность обеспечивалась за счет разработки правил движения. Затем появилась система

разметки дорог и регулирования перекрестков. И, наконец, стали строиться транспортные развязки, которые в принципе предотвращают пересечение потоков машин и пешеходов. Впрочем, используемые средства должны определяться характером решаемой задачи: для проселочной дороги вполне достаточно соблюдения простого правила - "смотри под ноги и по сторонам".

Основная идея структурного программирования: программа должна представлять собой множество блоков, объединенных в виде иерархической древовидной структуры, каждый из которых имеет один вход и один выход.

Любую программу можно построить, используя лишь три основных типа блоков:

1. функциональный блок - отдельный линейный оператор или их последовательность;
2. блок разветвления - If <условие> Then...Else.
3. обобщенный цикл - конструкция типа While <условие> Do <операторы> (проверка в начале цикла!);



Существенно, что каждая из этих конструкций имеет по управлению только один вход и один выход. Тем самым, и обобщенный оператор имеет только один вход и один выход.

Структурное программирование иногда называют еще "программированием без GO TO". Однако дело здесь не в операторе GO TO, а в его беспорядочном использовании. Очень часто при воплощении структурного программирования на некоторых языках программирования оператор перехода (GO TO) используется для реализации структурных конструкций, не снижая основных достоинств структурного программирования. Запутывают программу как раз "неструктурные" операторы перехода, особенно переход на оператор, расположенный в тексте модуля выше (раньше) выполняемого оператора перехода. Тем не менее, попытка избежать оператора перехода в некоторых простых случаях может привести к слишком громоздким структурированным программам, что не улучшает их ясность и содержит опасность появления в тексте модуля дополнительных ошибок. Поэтому можно рекомендовать избегать употребления оператора перехода всюду, где это возможно, но не ценой ясности программы.

К полезным случаям использования оператора перехода можно отнести выход из цикла или процедуры по особому условию, "досрочно" прекращающего работу данного цикла или данной процедуры, т.е. завершающего работу некоторой структурной единицы (обобщенного оператора) и тем самым лишь локально нарушающего структурированность программы. Большие трудности (и усложнение структуры) вызывает структурная реализация реакции на возникающие исключительные (часто ошибочные) ситуации, так как при этом требуется не только осуществить досрочный выход из структурной единицы, но и произвести необходимую обработку этой ситуации (например, выдачу подходящей диагностической информации). Обработчик исключительной ситуации может находиться на любом уровне структуры программы, а обращение к нему может производиться с разных нижних уровней. Вполне приемлемой с технологической точки зрения

является следующая "неструктурная" реализация реакции на исключительные ситуации. Обработчики исключительных ситуаций помещаются в конце той или иной структурной единицы и каждый такой обработчик программируется таким образом, что после окончания своей работы производит выход из той структурной единицы, в конце которой он помещен. Обращение к такому обработчику производится оператором перехода из данной структурной единицы (включая любую вложенную в нее структурную единицу).

Вообще говоря, главное в структурном программировании - грамотное составление правильной логической схемы программы, реализация которой языковыми средствами - дело вторичное.

CASE-средства (Computer - Aided Software Engineering) - это методы и технологии, которые позволяют проектировать различные информационные системы (в частности, базы данных) и автоматизировать их создание.

В зависимости от того, на каком этапе проектирования баз данных используются CASE-средства, их относят к:

CASE-средствам верхнего уровня. Их задействуют на начальных этапах проектирования, когда требуется выполнить анализ поставленной задачи, поставить цели и определить приоритеты, представить необходимую информацию в виде диаграмм и деревьев решений;

CASE-средствам нижнего уровня. С помощью этих средств выполняются заключительные этапы проектирования БД, проводятся собственно проектирование, написание кода, тестирование и внедрение программного обеспечения поддержки информационных систем.

интегрированным CASE-средствам, которые дают возможность выполнять все этапы проектирования БД благодаря наличию функций верхнего и нижнего уровней.

Основными характеристиками CASE средств, важными с точки зрения моделирования и оптимизации бизнес процессов, являются следующие:

Наличие графического интерфейса. Для представления моделей процессов CASE средства должны обладать возможностью отображать процессы в виде схем. Схемы много проще в использовании, чем различные текстовые и числовые описания. Это позволяет получать легко управляемые компоненты модели, обладающие простой и ясной структурой.

Наличие репозитория. Репозиторий это общая база данных, которая содержит описание элементов процессов и отношений между ними. Каждый объект репозитория должен обладать перечнем свойств, характерных только для этого объекта.

Гибкость применения. Эта характеристика дает возможность представлять бизнес процессы в различных вариантах, важных с точки зрения анализа. CASE средства должны позволять проводить анализ процессов и создавать модели, сфокусированные на различных аспектах деятельности предприятия.

Возможность коллективной работы. Анализ и моделирование процессов может требовать совместной работы нескольких человек. Для одновременной работы над моделями процессов CASE средства должны обеспечивать управление изменениями любыми фрагментами моделей и их модификацией при коллективном доступе.

Построение прототипов. Прототипы процессов необходимы для того, чтобы на ранних стадиях изменения процессов можно было понять, насколько процесс будет соответствовать требованиям.

Построение отчетов. CASE средства должны обеспечивать построение отчетов по всем моделям процессов с учетом взаимосвязи элементов. Такие отчеты необходимы для анализа моделей и определения возможностей по оптимизации. За счет отчетов обеспечивается контроль полноты и достаточности моделей, уровень декомпозиции процессов, правильность синтаксиса диаграмм и типов применяемых элементов.

Реинжиниринг программного обеспечения - это изучение и изменение системы для ее восстановления в новой форме. Принципы реинжиниринга применительно к процессу разработки программного обеспечения называются реинжинирингом программного обеспечения. Это положительно сказывается на стоимости программного обеспечения, качестве, обслуживании клиента и скорости доставки. В реинжиниринге программного обеспечения мы совершенствуем

программное обеспечение, чтобы сделать его более эффективным и эффективным.

Необходимость реинжиниринга программного обеспечения: Реинжиниринг программного обеспечения - это экономичный процесс разработки программного обеспечения и повышения качества продукта. Этот процесс позволяет нам определить бесполезное потребление развернутых ресурсов и ограничения, которые ограничивают процесс разработки, чтобы процесс разработки можно было сделать проще и экономически эффективным (время, финансы, прямое преимущество, оптимизация кода, косвенные выгоды и т.д.) и ремонтнопригодным. Реинжиниринг программного обеспечения необходим для-

a) Повышение производительности: реинжиниринг программного обеспечения повышение производительности за счет оптимизации кода и базы данных, что ускоряет обработку.

b) Непрерывность процессов: функциональность более старого программного продукта все еще может использоваться при тестировании или разработке программного обеспечения.

c) Возможность улучшения: Между тем процесс реинжиниринга программного обеспечения, не только качества программного обеспечения, функции и функциональность, но и ваши навыки совершенствуются, новые идеи попадают в ваш разум. Это заставляет разработчиков привыкать к захвату новых возможностей, чтобы можно было разрабатывать все новые и новые функции.

d) Снижение рисков: вместо разработки программного продукта с нуля или с начальной стадии здесь разработчики разрабатывают продукт с его существующей стадии для улучшения некоторых специфических функций, которые вызывают озабоченность заинтересованных сторон или его пользователей. Такая практика снижает вероятность ошибок.

e) Экономит время: как мы уже говорили выше, продукт разрабатывается с существующей стадии, а не с начальной, поэтому время, затрачиваемое на разработку программного обеспечения, меньше.

f) Оптимизация: этот процесс улучшает функции системы, функциональные возможности и снижает сложность продукта путем последовательной оптимизации как можно больше.

Дополнительные вопросы Структуры данных Графы

Нелинейные структуры данных: классификация Деревья

Использование деревьев в задачах поиска

Существует несколько возможных определений дерева. Например, с точки зрения теории графов деревом часто называют неориентированный граф с выделенной вершиной (корнем), который не содержит циклов. Нас будут интересовать не произвольные графы, а только ориентированные деревья, причем с точки зрения программистов. Поэтому мы используем следующее рекурсивное определение: дерево R с базовым типом T - это либо (а) пустое дерево (не содержащее ни одной вершины), либо (б) некоторая вершина типа T (корень дерева) с конечным (возможно, нулевым) числом связанных с ней деревьев с базовым типом T (эти деревья называются поддеревьями дерева R). Из этого определения, в частности, следует, что однонаправленный список (рисунок 10.1) является деревом.

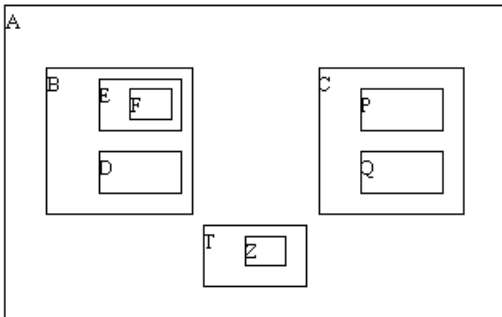


Рис. 10.1.

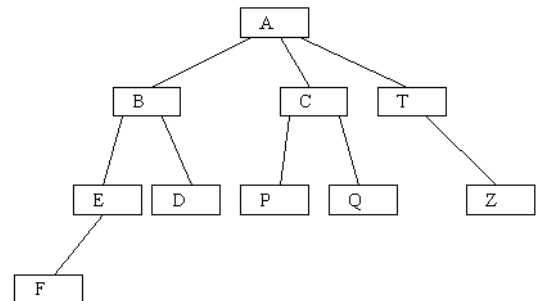


Рис.10.2.

Деревья можно представлять по-разному (это всего лишь однородная иерархическая структура). Например, на рисунках 10.1 и 10.2 показаны два разных способа представления одного и того же дерева, у которого базовый тип содержит множество букв латинского алфавита. Сразу заметим, что графовое представление на рисунке 2 больше соответствует специфике программирования.

Придерживаясь естественной для графового представления терминологии, мы будем называть связи между поддеревьями ветвями, а корень каждого поддерева - вершиной. **Упорядоченным деревом** называется такое, у которого ветви, исходящие из каждой вершины, упорядочены. Например, два упорядоченных дерева на рисунке 3 различаются.

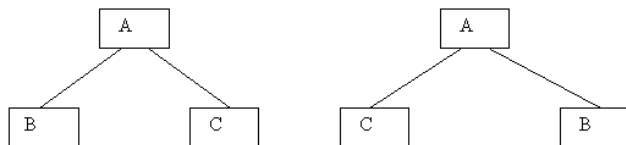


Рис.10.3.

По определению, корень дерева находится на уровне 0, а все вершины дерева, непосредственно связанные с вершиной уровня i , находятся на уровне $i+1$. Вершина x уровня i , непосредственно связанная с вершиной y уровня $i+1$, называется непосредственным предком (или родителем) вершины y . Такая вершина y соответственно называется непосредственным потомком (или сыном) вершины x . Вершина без непосредственных потомков называется листовой (или терминальной), нелистовые вершины называются внутренними. Под степенью внутренней вершины понимается число ее непосредственных потомков. Если все вершины имеют одну и ту же степень, то она полагается степенью дерева. На самом деле, всегда можно добиться того, чтобы любая вершина дерева имела одну и ту же степень путем добавления специальных вершин в тех точках, где отсутствуют поддеревья (рисунок 10.4).

Число вершин (или ветвей), которые нужно пройти от корня к вершине x , называется длиной пути к вершине x . Высотой (или глубиной) дерева будем называть максимальную длину его вершины.

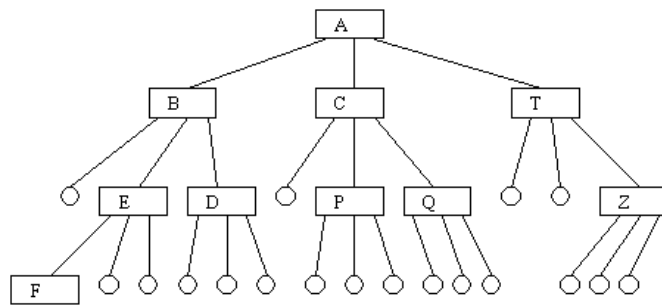


Рис.10.4.

Ориентированные, упорядоченные и бинарные
 Представление деревьев в памяти компьютера: последовательное и связанное
 размещение элементов
 Операции над деревьями
 Графы и их представление в компьютере
 Алгоритмы, оперирующие со структурами типа графа
 Файлы: организация и обработка, представление деревьями
 Алгоритмы поиска на графах
Работа с данными
 Проблема создания и сжатия больших информационных массивов,
 информационных хранилищ и складов данных
 Основные математические методы, применяемые при сжатии информации
 Фрактальные методы в архивации
 Управление складами данных