

17. Структура и состав вычислительной системы (аппаратура+программное обеспечение)

программным обеспечением. *Вычислительную систему* можно определить, как совокупность аппаратных и программных средств, функционирующих в единой системе и предназначенных для решения задач определенного класса. Рассмотрим структурную организацию вычислительной системы.

Традиционным представлением структуры вычислительной системы является пирамида (Рис. 4). Каждый из уровней пирамиды определяет свой уровень абстракции свойств вычислительной системы. Основанием является **аппаратный уровень** вычислительной системы — это основа всей иерархии, так как многие характеристики и функциональные возможности последующих программных уровней существенно определяются свойствами аппаратуры компьютера, находящегося в основе вычислительной системы.



Рис. 4. Структура вычислительной системы.

Представление о возможностях и свойствах конкретной вычислительной системы формируются с позиций каждого из уровней структурной организации. Так

Взаимосвязи уровней иерархической структуры вычислительной системы, их характеристики могут проявляться как в виде непосредственных межуровневых интерфейсов, определенных однозначным набором правил использования объектов одного уровня другим, так и косвенным влиянием одного уровня на другой. Примером остальных уровней, составляющими структурную организацию ВС.

Физические ресурсы (устройства) — компоненты аппаратуры компьютера, используемые на программных уровнях ВС или оказывающие влияние на функционирование всей ВС. Совокупность физических ресурсов составляет аппаратный уровень вычислительной системы.

Итак, аппаратный уровень вычислительной системы определяется набором аппаратных компонентов и их характеристик, используемых вышестоящими уровнями иерархии и оказывающих влияние на эти уровни. С позиций уровней, расположенных выше, аппаратный уровень предоставляют т.н. **физические ресурсы**, или **физические устройства** вычислительной системы. Каждому физическому ресурсу соответствуют определенные аппаратные компоненты компьютера и их характеристики. Физическими ресурсами являются процессор компьютера, оперативная память, внешние устройства, входящие в состав компьютера. Каждому физическому ресурсу вычислительной системы обычно соответствуют следующие характеристики:

- **правила программного использования**, определяющие возможность корректного использования данного ресурса в программе (для процессора компьютера эти правила описывают машинный язык — систему команд

данного компьютера, на основании которой возможно построение работающей программы, для внешнего устройства компьютера подобные правила описывают способы программного управления данным устройством, к примеру, это могут быть специальные команды ввода-вывода процессора);

- параметры физического ресурса, характеризующие его **объемные характеристики и/или производительность** (для процессора компьютера таким параметром может служить его тактовая частота, а для внешнего запоминающего устройства — объем информации, которая может храниться на данном устройстве и скорость доступа);
- степень использования данного физического ресурса в вычислительной системе — это параметры, которые характеризуют **степень занятости или используемости** данного физического ресурса (для процессора компьютера такой характеристикой является время его работы, затраченное на выполнение программ пользователей, для оперативного запоминающего устройства это будет объем используемой памяти, для линий связи — это ее загруженность).

1.1.3 Управление физическими ресурсами ВС

Уровень управления физическими ресурсами — это первый уровень **системного программного обеспечения** вычислительной системы. Его назначение — систематизация и стандартизация правил программного использования физических ресурсов. Для

Частичным решением этих проблем стало появление специальных стандартных программ — **драйверов физических ресурсов** (или **драйверов физических устройств**). *Драйвер физического устройства* — программа, основанная на использовании команд управления конкретного физического устройства и предназначенная для организации работы с данным устройством. Драйвер физического устройства скрывает от пользователя детальные элементы управления конкретным физическим устройством и предоставляет пользователю упрощенный программный интерфейс работы с устройством. Интерфейс драйвера физического устройства ориентирован на конкретные свойства устройства.

Совокупность драйверов физических устройств составляет уровень **управления физическими устройствами** вычислительной системы. Уровень управления физическими устройствами стандартизует правила, по которым возможно внесение в систему новых драйверов устройств. Следует отметить, что в системе для одного и того же физического устройства возможно наличие нескольких различных драйверов, которые имеют различные пользовательские интерфейсы, а также предоставляют различные возможности. Примером может служить устройство магнитной ленты, которое в Таким образом, на уровне управления физическими ресурсами (устройствами) вычислительной системы пользователю доступна система команд компьютера, а также интерфейсы драйверов физических устройств компьютера.

1.1.4 Управление логическими/виртуальными ресурсами

Развитием системного программного обеспечения стало появление уровня **управления логическими, или виртуальными, ресурсами** (или устройствами). В основу этого уровня легло обобщение особенностей физических устройств одного вида и создание драйверов, имеющих единые интерфейсы, посредством которых осуществляется доступ к различным физическим устройствам одного типа. Для этих целей в современных вычислительных системах предусмотрена возможность программного создания и использования т.н. *логических, или виртуальных, ресурсов* (виртуальное — нечто реально не существующее, не имеющее реальной, физической организации). *Логическое/виртуальное устройство (ресурс)* — это устройство/ресурс, некоторые эксплуатационные характеристики которого (возможно все) реализованы программным образом. Современные вычислительные системы позволяют создавать разнообразные логические/виртуальные устройства и соответствующие им драйверы. *Драйвер логического/виртуального ресурса* — это программа, обеспечивающая существование и использование соответствующего ресурса. Для этих целей при его реализации возможно использование существующих драйверов физических и виртуальных устройств.

Основным результатом появления уровня управления виртуальными устройствами вычислительной системы стала многоуровневая унификация интерфейсов доступа к ресурсам вычислительной системы, что существенно упростило проблему программирования устройств компьютера, а также предоставило качественно новые возможности в функционировании вычислительных систем и в создании их программного обеспечения. Примером могут служить файловые системы, которые обеспечивают простые и надежные интерфейсы именованного хранения и использования данных, полностью скрывая от пользователя проблемы ее внутренней организации. К примеру, пользователь современной вычислительной системы может не только не знать, на каком внешнем запоминающем устройстве размещены данные его файлов, он может не знать и территориальное расположение и тип компьютера, на котором хранятся его данные. Существенное развитие получили средства **управления виртуальными устройствами (ресурсами)**, которые обеспечивают контроль за созданием и использованием ресурсов вычислительной системы.

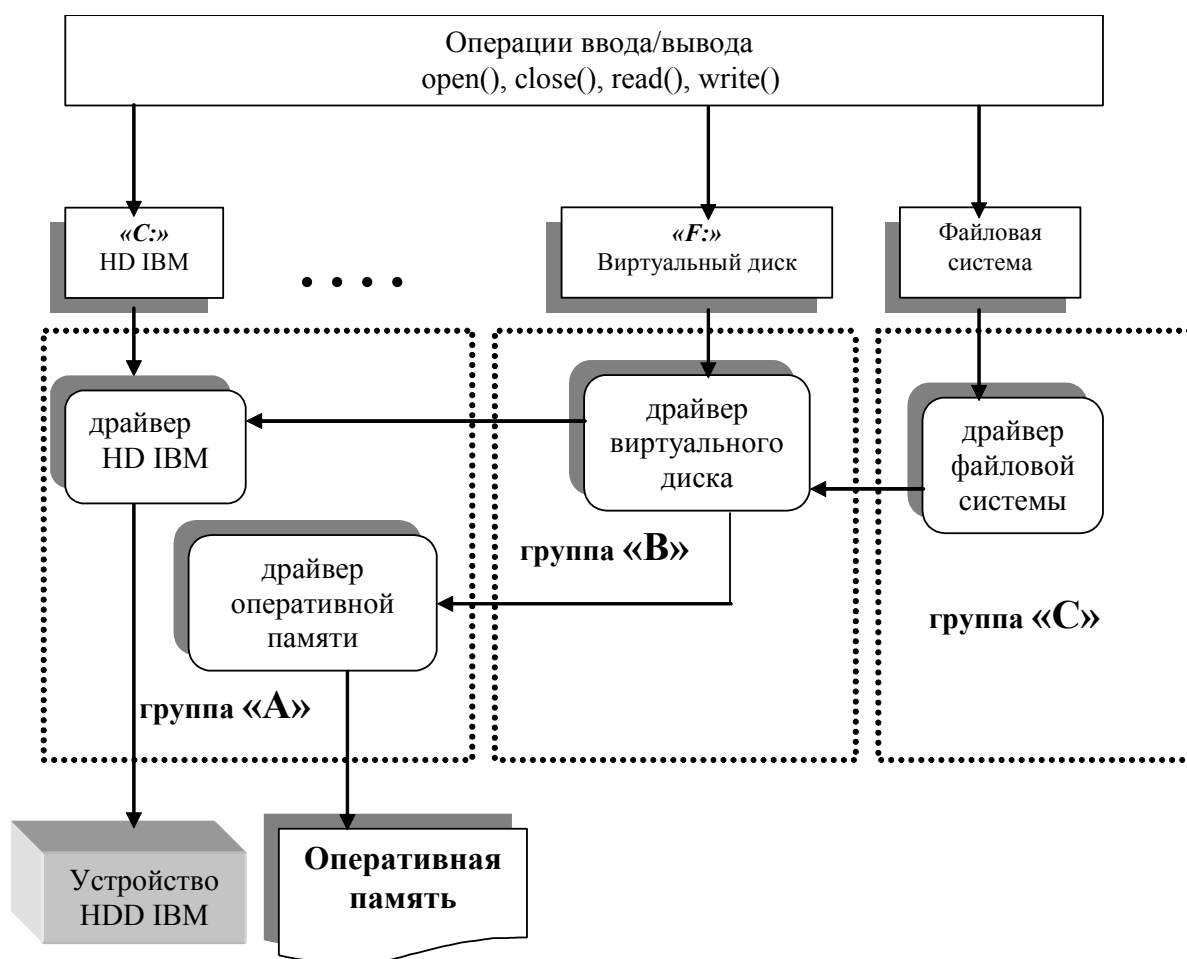


Рис. 6. Схема организации ввода-вывода в системе.

системе. Под **ресурсами вычислительной системы** мы будем понимать совокупность всех физических и виртуальных ресурсов. Одной из характеристик ресурсов вычислительной системы является их конечность. То есть рано или поздно в системе

возникает конкуренция за обладание ресурсом между его программными потребителями. При этом если речь идет о таком виртуальном ресурсе, как файловая система, то конечным является размер файловой системы на устройствах хранения данных, ограничения на предельное количество зарегистрированных в файловой системе файлов. Именно за эти параметры возможно возникновение конкуренции при использовании файловой системы. А теперь попытаемся вернуться к проблеме определения понятия операционной системы. **Операционная система** — это комплекс программ, обеспечивающий управление ресурсами вычислительной системы. Это основная

С точки зрения средств программирования, доступных на уровне управления виртуальными ресурсами, пользователю предоставляются система команд компьютера, а также интерфейсы, обеспечивающие доступ к устройствам компьютера (как физическим, так и виртуальным). Доступная пользователю совокупность интерфейсов устройств компьютера может включать в себя как аппаратные интерфейсы доступа к устройствам, так и драйверы физических и/или виртуальных устройств. Конкретный состав интерфейсов определяется свойствами вычислительной системы, соответствующими, уровнями управления ресурсами, а также привилегиями пользователя (об этом подробнее мы будем говорить несколько позднее).

Вернемся к рассмотрению следующего уровня иерархической организации вычислительных систем — к **уровню систем программирования**. **Система программирования** — комплекс программ, обеспечивающий поддержание этапов **жизненного цикла программы в вычислительной системе**. Этапы жизненного цикла

1.1.5 Системы программирования

Прежде чем начать рассматривать следующий уровень структурной организации вычислительных систем, обратимся к последовательности этапов, традиционно связываемых с разработкой и внедрением программных систем. Совокупность этих этапов составляют **жизненный цикл программы** в вычислительной системе. Остановимся на основных задачах, решаемых на каждом из этапов жизненного цикла программы. Следует

Проектирование программной системы. На данном этапе принимаются решения, традиционно включающие в себя следующие шаги.

- Исследование решаемой задачи, формирование концептуальных требований к разрабатываемой программной системе.
- Определение характеристик **объектной вычислительной системы** — характеристик аппаратных и программных компонентов вычислительной системы, в рамках которой будет работать создаваемая программная система.
- Построение моделей функционирования автоматизируемого объекта.
- Определение характеристик **инструментальной вычислительной системы** — вычислительной системы, которая будет использоваться при создании программной системы. Зачастую характеристики объектной и инструментальной вычислительной системы совпадают: тип вычислительных систем, на которых в дальнейшем будет работать программная система, совпадает с типом вычислительной системы, которая использовалась при разработке. Однако, в общем случае это не совсем так. Тип и
- Выбор основных алгоритмов, инструментальных средств, которые будут использованы при программировании, а также разработка архитектуры программного решения, включающей разбиение программного решения на основные модули и определение информационных связей между модулями системы, а также правила взаимодействия с объектной вычислительной системой.
- Априорная оценка ожидаемых результатов. Один из важнейших шагов проектирования программной системы, заключающийся в предварительной оценке характеристик проектируемого решения до начала его практической реализации. Для проектирования разбивается на линейную последовательность этих шагов. Часто проектирование представляет собою итерационный процесс, в котором возможны неоднократные возвраты к тем или иным шагам (Рис. 7).

Следующий этап жизненного цикла программы — **кодирование (программная реализация, или реализация)**. Это этап построения кода программой системы на

основании спецификаций, полученных при ее проектировании. На данном этапе используются инструментальные средства программирования:

- трансляторы языков программирования, средства поддержки и использования библиотек программ, формирования модулей, которые могут исполняться в вычислительной системе;
- средства управления разработкой программных продуктов коллективом разработчиков.

Результатом этапа кодирования является реализация программной системы, которая может представляться в виде совокупности исходных модулей программы, объектных или библиотечных модулей, а также модулей исполняемого кода разрабатываемой программной системы (Рис. 8). Большое значение для разработки

больших, логически сложных программных систем имеют средства управления разработкой программных продуктов, которые позволяют организовать эффективную коллективную работу над реализацией программного проекта. Традиционно они включают в себя следующие компоненты:

- средства автоматизации контроля использования межмодульных интерфейсов,
- средства автоматизации получения объектных и исполняемых модулей программы,
- система поддержки версий — система, позволяющая фиксировать состояние

Этап **тестирования и отладки** программной системы. Можно представить программу в виде некоторого автомата, получающего на входе исходные данные, а на выходе формирующий результат (Рис. 9). Одной из задач проектирования программной системы является определение ее правил функционирования, точнее, правил, по которым для входных данных формируются выходные данные (или результаты). Тестирование программы — процесс проверки правильности функционирования программы на заранее определенных наборах входных данных — **тестах**, или **тестовых нагрузках**. В общем представляется возможным. Поэтому актуальной задачей в тестировании является решение проблемы формирования минимального набора тестов или тестовых нагрузок, наиболее полно проверяющих функциональность программы (**тестовое покрытие**).

Другим компонентом данного этапа является **отладка**. Отладка — это поиск, локализация и исправление зафиксированных при тестировании или в процессе эксплуатации ошибок. Для обеспечения процесса отладки используются специальные программные средства — отладчики. Средства отладки существенно зависят от типа и назначения создаваемой программной системы.

Этап **ввода программной системы в эксплуатацию (внедрение) и сопровождения**. Немаловажным этапом жизненного цикла программы в вычислительной системе является этап, связанный с представлением разрабатываемой программной системы в качестве программного продукта. Одним из основных требований, предъявляемых к программному продукту, является возможность эксплуатации соответствующей программной системы без постоянного участия разработчика программы. Это достигается, с одной стороны, соответствующей надежностью программы (для этого программа должна быть максимально полно протестирована и устойчива к всевозможным комбинациям входных данных), а с другой стороны — это наличие подробной и адекватной программе документации, необходимой для всех категорий пользователей данной программной системы (пользователь, системный программист, администратор, оператор и т.п.).

Современные технологии разработки программного обеспечения специфицируют различные модели организации жизненного цикла программной системы. Традиционная

модель — **каскадная модель** (Рис. 10) — представляет разработку в виде строго линейной последовательности этапов, каждый из которых заканчивается фиксацией результата, и только после этого начинается последующий.



Рис. 10. Каскадная модель.

В определенном смысле эта модель является вырожденной, т.к. соблюсти эти правила на практике достаточно сложно. Примером может служить связка этапа тестирования и отладки с предшествующими этапами, которая по своей сути итерационна (после обнаружения и локализации ошибки зачастую необходимо вернуться к этапу кодирования, а возможно и проектирования). Прагматическим развитием каскадной модели является **каскадная итерационная модель** (Рис. 11), которая в общем случае, предоставляет возможность осуществления анализа полученных на этапе результатов и возврат к любому предшествующему этапу.

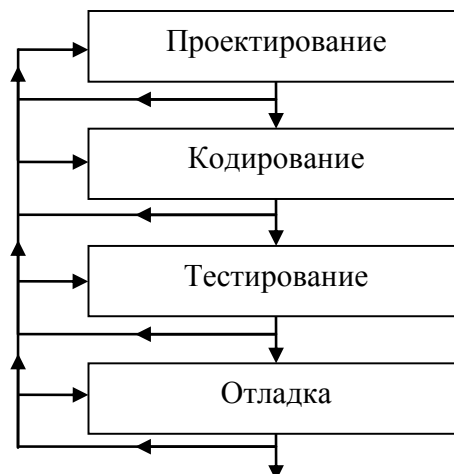


Рис. 11. Каскадная итерационная модель.

Современные технологии разработки программного обеспечения помимо каскадной модели используют и другие модели организации жизненного цикла программных систем. В частности, популярной является **спиральная модель** организации жизненного цикла (Рис. 12).

Данная модель основана на том, что процесс разработки программной системы складывается из последовательности "спиралей", каждая из которых включает этапы проектирования, кодирования, тестирования и получения результата. Под результатом

понимается очередная детализация проекта и получение последовательности программ — **прототипов**. Прототип — программа, реализующая частичную функциональность и внешние интерфейсы разрабатываемой системы. Последовательность прототипов, в конечном счете, сходится к реализации программной системы. А детализации проекта, в итоге, превращаются в полный проект системы.

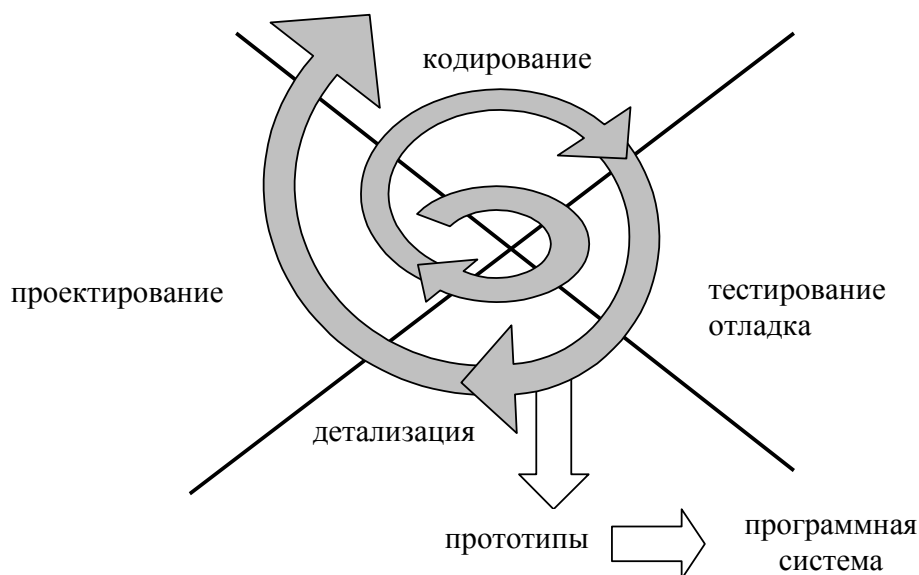


Рис. 12. Спиральная модель.

«система программирования — это комплекс программ, обеспечивающий технологию автоматизации проектирования, кодирования, тестирования, отладки и сопровождения программного обеспечения». Функции конкретной системы программирования

Итак, мы переходим к вершине структурной организации вычислительных систем — к уровню **прикладного программного обеспечения**. **Прикладная система** — это программная система, ориентированная на решение или автоматизацию решения задач из конкретной предметной области. Прикладная система является прагматической основой всей вычислительной системы, так как, в конечном счете, именно для решения конкретных прикладных задач создавались все те уровни вычислительной системы, которые мы рассмотрели к настоящему времени.

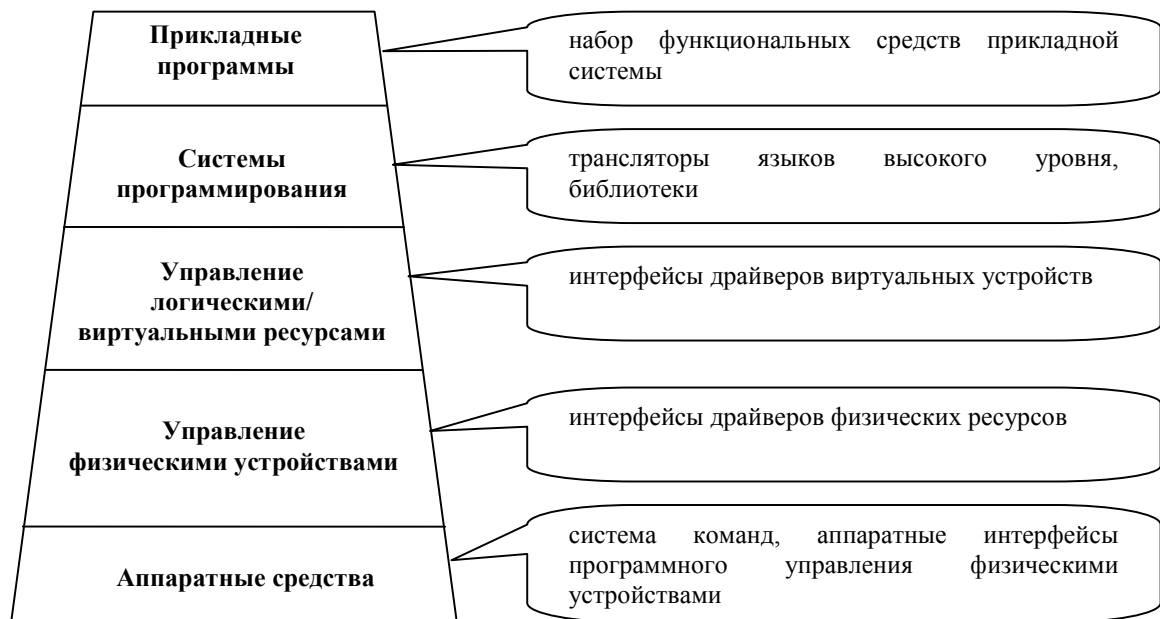
Современная прикладная система предполагает глубокую интеграцию всех компонентов вычислительной системы: аппаратной части, операционной системы, системы программирования. В итоге, возможно разделение пользователей прикладной системы на следующие категории:

- **оператор** или **прикладной пользователь**, оперируя средствами пользовательского интерфейса и функциональными возможностями системы, решает конкретные прикладные задачи. Примером может служить работа инженера по проектированию оборудования с использованием системы AutoCAD или работа менеджера крупной компании, использующей аналитические средства системы управления бизнесом на основе решений BAAN;
- **системный программист** — пользователь компонентов прикладной системы, обеспечивающий возможности интеграции данной системы в конкретной вычислительной системе, возможности настройки в соответствии с конкретными особенностями эксплуатации системы на конкретном предприятии, доработку функциональных возможностей системы, удовлетворяющих потребностям и особенности эксплуатации. Например, применение пакета Microsoft Office с точки

- **системный администратор** обеспечивает выполнение текущих работ по поддержке функционирования программной системы в конкретных условиях: в их состав могут входить регистрация пользователей и распределение полномочий и прав между ними, контроль за обеспечением сохранности и целостности данных, фиксация проблем,

возникающих в процессе эксплуатации, и обоснованное выполнение обновлений системы, поступающих от разработчика.

Каждой категории пользователей прикладной системы предоставлены свои, специализированные средства работы, которые предназначены для решения конкретных задач данного пользователя.



и внутренней организации данных реального компьютера.

Уровень прикладных систем. Тенденция "расширения" возможностей компьютера продолжается и на прикладном уровне. При этом для каждой категории пользователей прикладного уровня вычислительной системы существует свое расширение компьютера. Так, например, для оператора прикладной системы компьютер представляется набором функциональных средств прикладной системы, доступной через пользовательский интерфейс. Рассмотрим работу кассира в современном супермаркете, кассовый аппарат которого может являться специализированным персональным компьютером, работающим в составе системы автоматизации деятельности всего магазина. Для кассира работа с этим компьютером и, соответственно, возможности этого компьютера представляются в виде возможностей прикладной подсистемы, автоматизирующей его рабочее место. Заведомо кассир магазина может не иметь никаких представлений о внутренней организации специализированной вычислительной системы, на которой он работает (тип компьютера, тип операционной системы, состав драйверов ОС и т.п.).

образуются за счет соответствующих уровней вычислительной системы. Мы будем говорить, что каждый пользователь, работая в соответствующем расширении компьютера, работает в **виртуальной машине** или **виртуальном компьютере**. Реальный компьютер используется непосредственно исключительно на аппаратном уровне. Во всех остальных случаях пользователь работает с программным расширением возможностей реального компьютера — с виртуальным компьютером. Причем "виртуальность" этого компьютера (или этих компьютеров) возрастает от уровня управления физическими ресурсами ВС до уровня прикладных систем.

18. Основные компоненты архитектуры ЭВМ (процессор, устройства памяти, внешние устройства).

Основные из принципов построения ЭВМ сформулированные Джоном фон Нейманом

Принцип линейности и однородности памяти. *Память* машины фон Неймана — это линейная (упорядоченная) и однородная последовательность некоторых элементов, называемых *ячейками*. В любую ячейку памяти другие устройства машины (по толстым стрелкам на рис. 2.1) могут записать и считать информацию, причем время чтения из любой ячейки одинаково для всех ячеек. Время записи в любую ячейку тоже одинаково (это и есть принцип *однородности* памяти). Разумеется, время чтения из ячейки памяти может не совпадать со временем записи в нее. Такая память в современных компьютерах называется *памятью с произвольным доступом* (Random Access Memory — RAM). На практике многие современные ЭВМ могут иметь участки памяти разных видов. Например, некоторые области памяти поддерживают только чтение информации (по-английски эта память называется Read Only Memory — ROM), данные в такую память записываются один раз при изготовлении этой памяти. Другие области памяти могут допускать запись, но за значительно большее время, чем в обычную память (это так называемая *полупостоянная* память, такой памятью комплектуются популярные в настоящее время карты флэш-памяти) и др.

Ячейки памяти в машине фон Неймана нумеруются от нуля до некоторого положительного числа N (это и означает, что память *линейная*), причем число N в «настоящих» ЭВМ часто является степенью двойки минус единица. *Адресом ячейки* называется ее номер. Каждая ячейка состоит из более мелких частей, именуемых *разрядами* и нумеруемых также от нуля и до определенного числа. Количество разрядов в ячейке обозначает *разрядность памяти*. Каждый разряд может хранить одну *цифру* в некоторой системе счисления. В большинстве ЭВМ используется двоичная система счисления, так как это более выгодно с точки зрения аппаратной реализации. В этом случае каждый разряд хранит одну двоичную цифру или один *бит* инфор-

мации. Восемь бит составляют один *байт*. Сам фон Нейман тоже был сторонником использования двоичной системы счисления, что позволяло хорошо описывать архитектуру узлов ЭВМ с помощью логических (булевских) выражений.

Содержимое ячейки называется *машинным словом*. С точки зрения архитектуры, машинное слово — это минимальный объем данных, которым могут обмениваться между собой различные узлы машины по толстым стрелкам на схеме (не надо, однако, забывать о передаче сигналов по тонким стрелкам). Из каждой ячейки памяти можно считать *копию* машинного слова и передать ее в другое устройство компьютера, при этом оригинал не меняется. При записи в память старое содержимое ячейки пропадает и заменяется новым машинным словом.

Принцип неразличимости команд и данных. С точки зрения программиста машинное слово представляет собой либо команду, либо подлежащее обработке данное (это число, символьная информация, элемент изображения и т.д.). Для краткости в дальнейшем будем называть такую информацию «числами». Данный принцип фон Неймана заключается в том, что числа и команды *неотличимы* друг от друга — в памяти и те и другие представляются некоторым набором разрядов, причем по внешнему виду машинного слова нельзя определить, что оно собой представляет — команду или число.

Из неразличимости команд и данных вытекает очевидное следствие — **принцип хранимой программы**. Этот принцип является очень важным, его суть состоит в том, что программа хранится в памяти вместе с числами. Чтобы понять важность этого принципа, рассмотрим компьютеры (и компьютеры). А теперь поймем, что на вход некоторого алгоритма можно, вообще говоря, в качестве входных данных подавать запись некоторого другого алгоритма (в частности, свою собственную запись). Остается сделать последний шаг в этих рассуждениях и понять, что *выходными* данными алгоритма тоже может быть запись некоторого другого алгоритма. Таким образом, одна программа может в качестве результата своей работы поместить в память компьютера другую программу. Как вы уже вероятно знаете, именно так и работают *компиляторы*, переводящие программы с одного языка на другой.

Следствием принципа хранимой программы является то, что программа может *изменяться* во время счета самой этой программы. Говорят также, что программа может *самомодифицироваться*, т.е. изменять сама себя во время своего счета. В настоящее время само-

Принцип автоматической работы. Этот принцип еще называют принципом **программного управления**. Машина, выполняя записанную в ее памяти *программу*, функционирует автоматически, без участия человека, если только такое участие не предусмотрено в самой программе, например, при вводе данных с клавиатуры. Пример устройства, которое может выполнять команды, как и ЭВМ, но не в автоматическом режиме — обычный (непрограммируемый) калькулятор.

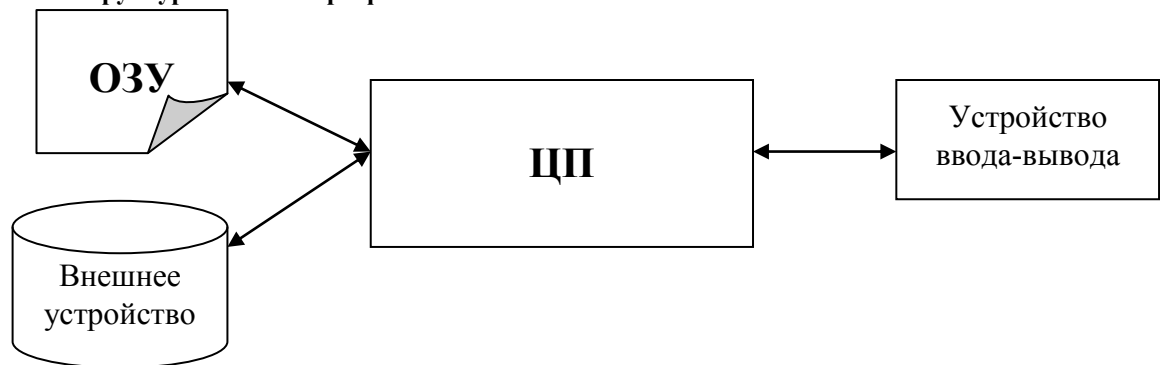
Программа — набор записанных в памяти (не обязательно последовательно) машинных команд, задающих *операции*, описывающих шаги работы алгоритма. Команды программы обрабатывают хранимые

Принцип последовательного выполнения команд. Устройство управления выполняет некоторую команду *от начала до конца*, а затем по определенному правилу выбирает следующую команду для выполнения, затем следующую и т. д. При этом каждая команда либо сама явно указывает на команду, которая будет выполняться следующей (такие команды называются командами перехода), либо следующей будет выполняться команда из ячейки, расположенной в памяти непосредственно вслед за той ячейкой, в которой хранится только что выполненная команда. Этот процесс продолжается, пока не будет выполнена специальная команда останова, либо при выполнении очередной команды не возникнет *аварийная ситуация* (например,

1. **Принцип двоичного кодирования информации:** вся информация, которая поступает и обрабатывается в компьютере, кодируется в двоичной системе счисления.

Рассмотрим упрощенную структуру компьютера фон Неймана (Рис. 20):

Рис. 20. Структура компьютера фон Неймана.



– **Оперативное запоминающее устройство (ОЗУ), или основная память,** — устройство хранения данных, в котором находится исполняемая в данный момент программа. То есть оперативная память — это свойство всё-таки системное, а не технологическое (т.е. на основе чего сделано это устройство — на той же элементной базе можно сделать устройство для хранения информации, но оно будет представляться системе не как оперативная память, а как внешнее устройство; тогда как программа будет исполняться из оперативной памяти).

– **Внешние устройства** — программно управляемые устройства, входящие в состав компьютера, т.е. устройства, с которыми выполняемая программа может осуществлять обмен данными.

– **Процессор, или центральный процессор (ЦП),** — основной компонент компьютера, обеспечивающий выполнение программ, процессор координирует работу внешних устройств и оперативной памяти. Процессор состоит из **арифметико-логического устройства (АЛУ)** и **устройства управления (УУ)**. Устройство управления

обеспечивает последовательную выборку команд, составляющих программу, из памяти, выделение и анализ кода операции, получение значений операндов. В зависимости от кода операции команда выполняется либо в устройстве управления (обычно это могут быть команды передачи управления), либо код операции и операнды передаются для выполнения в АЛУ. После чего выбирается из памяти следующая команда программы и т.д. В системе команд процессора предусмотрены средства для взаимодействия с внешними устройствами.

Устройство управления тоже имеет свои регистры, оно может считывать команды из памяти на специальный *регистр команд* РК (instruction register — IR), на котором всегда хранится *текущая* выполняемая команда. Регистр УУ с именем РА называется *регистром* или *счетчиком адреса* (в англоязычной литературе его часто обозначают IP — instruction pointer), при выполнении текущей команды в него по определенным правилам записывается адрес *следующей* выполняемой команды (первую букву в сокращении слова «регистр» мы будем в дальнейшем изложении часто записывать латинской буквой R).

Опишем теперь более формально шаги выполнения одной команды в машине фон Неймана:

- 1) $РК := <РА>$; считать из ячейки памяти с адресом РА команду на регистр команд РК;
- 2) $РА := РА + 1$; увеличить счетчик адреса на единицу;
- 3) выполнить очередную команду, хранящуюся в регистре РК.

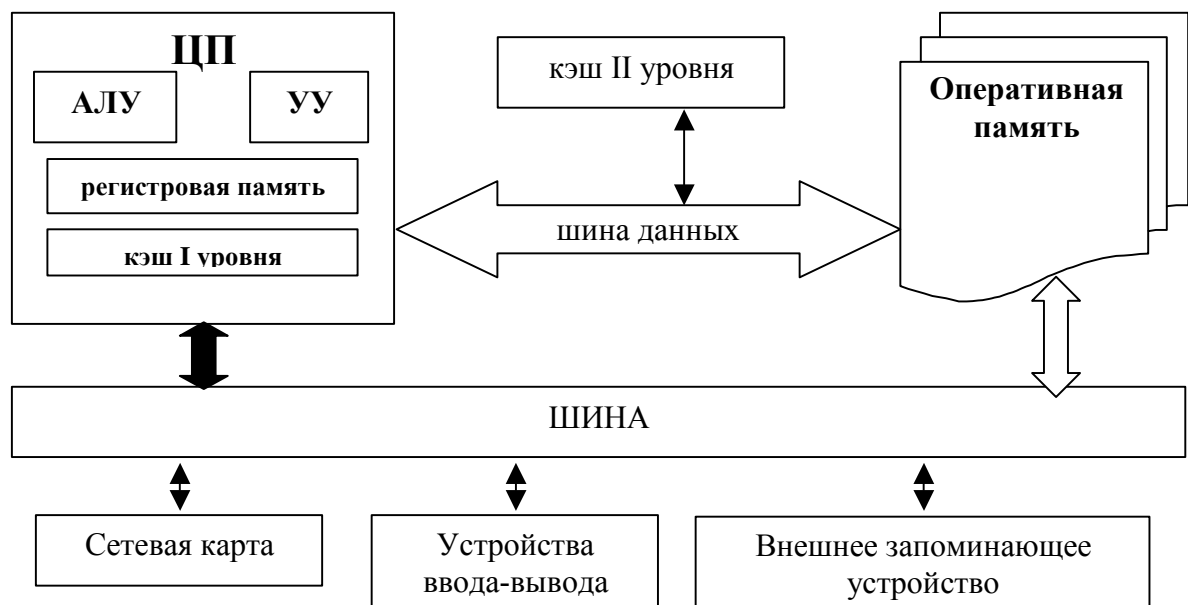


Рис. 21. Базовая архитектура современных компьютеров.

1.2.2 Оперативное запоминающее устройство

Оперативное запоминающее устройство (RAM — Random-Access Memory, память с произвольным доступом) — это устройство для хранения данных, в котором находится исполняемая программа. ОЗУ еще называют основной памятью, или оперативной памятью. Команды программы, исполняемые компьютером, поступают в процессор исключительно из ОЗУ. Таким образом, основным назначением оперативной памяти является хранение программы, которая выполняется в настоящее время компьютером. Оперативная память состоит из **ячеек памяти**. Ячейка памяти — это устройство, в котором размещается информация. Ячейка памяти может состоять из двух полей (Рис. 22). Первое поле — поле машинного слова, второе — поле служебной информации (или ТЭГ).

Машинное слово — поле программно изменяемой информации. В машинном слове могут располагаться машинные команды (или части машинных команд) или данные, с которыми может оперировать программа. Машинное слово имеет фиксированный для данной ЭВМ размер. Обычно под размером машинного слова понимается количество двоичных разрядов, размещаемых в машинном слове. Когда используются термины «16-ти разрядный компьютер», или «32-х разрядный компьютер», или «64-х разрядный компьютер», это означает, что речь идет о компьютерах, оперативная память которых имеет *машинные слова* размером 16, 32 или 64 разряда соответственно.

Поле служебной информации — ТЭГ (tag — ярлык, бирка) — поле ячейки памяти, в котором схемами контроля процессора и ОЗУ автоматически размещается информация, необходимая для осуществления контроля за целостностью и корректностью использования данных, размещаемых в соответствующем машинном слове. В принципе ТЭГ может отсутствовать.

Использование поля служебной информации (ТЭГа) может осуществляться в следующих целях.

- **Контроль за целостностью данных.** Простейшая модель — это одноразрядный ТЭГ, который использовался для контроля чётности. Содержимое поля

- **Контроль доступа к командам/данным.** заключается в следующем. Когда мы размещаем программу в памяти и генерируем код, то мы сразу же «раскрашиваем» всю информацию на два цвета — машинные слова, в которых находятся команды, и машинные слова, в которых находятся данные. При включении

- **Контроль доступа к машинным типам данных.** соответствующего типа. Контроль доступа к машинным типам данных осуществляется за счет фиксации в поле ТЭГа кода типа данных при их записи в машинное слово, а при использовании этих данных в качестве операндов команд осуществляется автоматическая проверка совпадения типа операнда и типа команды. Если они совпадают, то команда продолжает свое выполнение, если нет, то происходит прерывание. Как видим, контроль

Наличие или отсутствие поля служебной информации в ячейке памяти, характер его использования зависят от конкретного типа компьютеров. В каких-то компьютерах это поле ячейки памяти может отсутствовать, и в этом случае размер ячейки памяти совпадает с машинным словом. В каких-то — поле со служебной информацией ячейки памяти есть и используется для организации контроля за целостностью данных и корректностью их использования.

В ОЗУ все ячейки памяти имеют уникальные имена, имя — **адрес ячейки памяти**. Обычно адрес — это порядковый номер ячейки памяти (нумерация ячеек памяти возможна как подряд идущими номерами, так и номерами, кратными некоторому значению). Доступ к содержимому машинного слова осуществляется при непосредственном (например, считать содержимое слова с адресом **А**) или косвенном

использовании адреса (например, считать значение слова, адрес которого находится в машинном слове с адресом **В**). Одной из характеристик оперативной памяти является ее производительность, которая определяет скорость доступа процессора к данным, размещенным в ОЗУ. Обычно производительность ОЗУ определяется по значениям двух параметров.

- Первый — **время доступа (access time — t_{access})** — это время между запросом на чтение слова из оперативной памяти и получением содержимого этого слова.

- Второй параметр — **длительность цикла памяти (cycle time — t_{cycle})** — это минимальное время между началом текущего и последующего обращения к памяти.

Обычно, длительность цикла памяти существенно превосходит время доступа ($t_{\text{cycle}} > t_{\text{access}}$). Это связано с тем, что устройства памяти устроены таким образом, что после операции чтения многие из устройств памяти требуют регенерации (т.е. при чтении информация из ячейки разрушается – для того, чтобы сохранить информацию, надо её записать). Реальные соотношения между длительностью цикла и временем доступа зависят от конкретных технологий, применяемых для организации ОЗУ (в некоторых ОЗУ $t_{\text{cycle}}/t_{\text{access}} > 2$). Последнее утверждение говорит о том, что возможна ситуация, при которой для чтения N слов из памяти потребуется времени больше, чем $N \times t_{\text{access}}$.

Расслоение ОЗУ — один из аппаратных путей решения проблемы дисбаланса в скорости доступа к данным, размещенным в оперативной памяти, и производительностью процессора. Суть расслоения состоит в следующем (Рис. 24, Рис. 25).

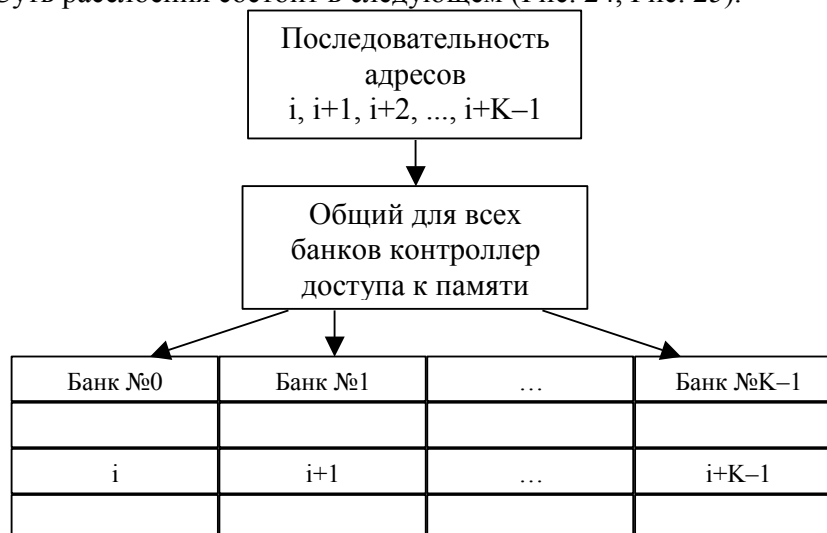


Рис. 24. Централизованный контроллер доступа к ОЗУ — один контроллер на все банки.

Всё пространство ОЗУ делится на **K независимых** подустройств, которые называются *банками памяти*. Обычно размер каждого из банков – это некоторая степень двойки ($K = 2^L$). Адресация в системе организована таким образом, что младшие L разрядов адреса содержат номер банка (Рис. 25). Подряд идущие ячейки памяти распределены между банками таким образом, что у любой ячейки ее соседи размещаются

в соседних банках (т.е. они находятся в разных банках). Что дает подобная организация памяти? Расслоение памяти позволяет во многом сократить задержки, возникающие из-за несоответствия времени доступа и цикла памяти при выполнении последовательного доступа к ячейкам памяти, т.к. при расслоении ОЗУ задержки, связанные с циклом памяти, будут возникать только в тех случаях, когда подряд идущие обращения попадают

в один и тот же банк памяти. Таким образом, расслоение памяти позволяет добиться увеличения скорости чтения из памяти при последовательном доступе. Используя организацию параллельной работы банков, в идеальном случае, можно повысить производительность работы ОЗУ в K раз. Для этих целей необходимо использовать более сложную архитектуру системы управления памятью.

Возможны две модели организации доступа к памяти – с *централизованным контроллером доступа к памяти* (Рис. 24 – один контроллер управляет всеми банками; в этом случае нет проблемы цикла памяти, т.к. соседние ячейки памяти находятся в разных банках; но нет эффекта при параллелизме) и с *контроллерами для каждого из банков* (Рис. 25 – в этом случае мы можем организовывать параллельный доступ к памяти, т.е. одновременно мы можем считать порцию данных до K слов).

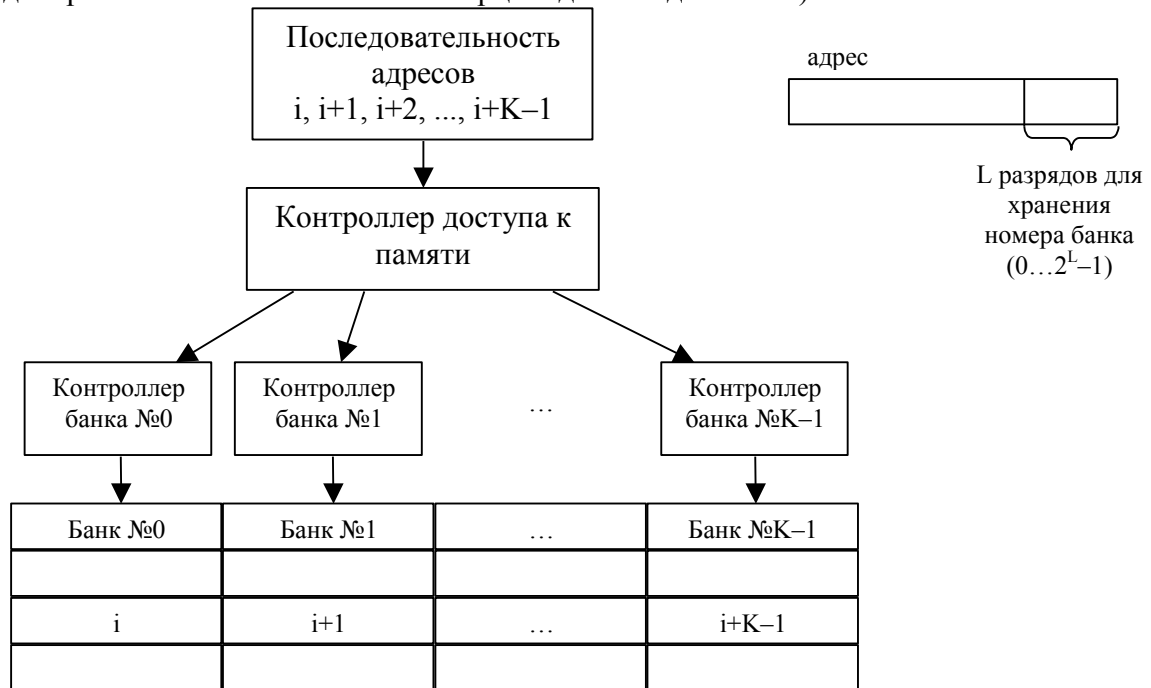


Рис. 25. ОЗУ с расслоением памяти — каждый банк обслуживает отдельный контроллер.

1.2.3 Центральный процессор

Процессор, или **центральный процессор (ЦП)**, компьютера обеспечивает выполнение машинных команд, составляющих программу, размещенную в оперативной памяти. Термин «центральный процессор» соответствует ситуации сегодняшнего дня,

Рассмотрим основные компоненты обобщенной структурной организации центрального процессора (Рис. 26).

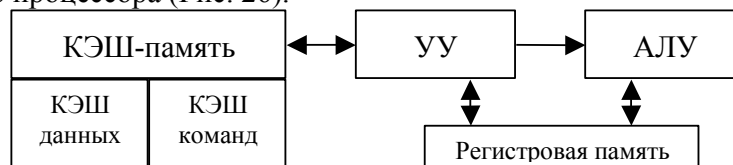


Рис. 26. Структура организации центрального процессора.

1.2.3.1 Регистровая память

1.2.3.1 Регистровая память

Регистровая память, или **регистровый файл** (register file), — совокупность устройств памяти процессора (так называемых *регистров*), предназначенных для временного хранения управляющей информации, операндов и/или результатов выполняемых команд. Регистровая память обычно включает в себя **регистры общего назначения** (general-purpose register) и **специальные регистры** (special-purpose register).

Регистры общего назначения (РОН) состоят из доступных для программ пользователей регистров, предназначенных для хранения операндов, адресов операндов, результатов выполнения команд. РОН могут иметь машинную типизацию (например, регистры для хранения данных с плавающей точкой, с фиксированной точкой и т.д.). РОН могут быть скалярными (когда с одним регистром ассоциируется только одна единица памяти) и векторными (например, с одним регистром может ассоциироваться вектор регистров из 64 элементов; примером классических векторных компьютеров являются компьютеры фирмы CRAY). Для чего нужны РОН? Регистровая память работает в темпе процессора, т.е. скорость доступа к содержимому регистров сравнима со скоростью обработки информации процессором, поэтому одной из основных причин появления регистров общего назначения было сглаживание дисбаланса в производительности процессора и скорости доступа к оперативной памяти. РОН были первым аппаратным средством, которое предоставлялось пользователю для оптимизации своей программы.

Специальные регистры предназначены для координации информационного взаимодействия основных компонентов процессора. В их состав могут входить специальные регистры, обеспечивающие управление устройствами компьютера, регистры, содержимое которых используется для представления информации об актуальном состоянии выполняемой процессором программы и т.д. Так же, как и в случае регистров конкретного процессора. К наиболее распространенным специальным регистрам относятся: **счетчик команд** (program counter), **указатель стека** (stack pointer), **слово состояния процессора** (processor status word). **Счетчик команд** — специальный регистр, в котором размещается адрес очередной выполняемой команды программы. Счетчик команд изменяется в устройстве управления согласно алгоритму, заложенному в программу. Более подробно использование счетчика команд проиллюстрируем несколько позднее при рассмотрении рабочего цикла процессора. **Указатель стека** — регистр, содержимое которого в каждый момент времени указывает на адрес слова в области памяти, являющегося вершиной стека. Обычно данный регистр присутствует в процессорах, система команд которых поддерживает работу со стеком (операции чтения и записи данных из/в стек с автоматической коррекцией значения указателя стека). **Слово состояния процессора** — регистр, содержимое которого определяет режимы работы процессора, значения кодов результата операций и т.п.

1.2.3.2 Устройство управления. Арифметико-логическое устройство

Устройство управления (control unit) — устройство, которое координирует выполнение команд программы процессором. **Арифметико-логическое устройство** (arithmetic/logic unit) обеспечивает выполнение команд, предусматривающих арифметическую или логическую обработку операндов. Эти устройства являются своего рода «мозгом» процессора, т.к. именно функционирование устройства управления арифметико-логического устройства обеспечивают выполнение программы. Рассмотрим упрощенную (без анализа нештатных ситуаций) схему выполнения процесса программы (Рис. 27) в модельном компьютере.

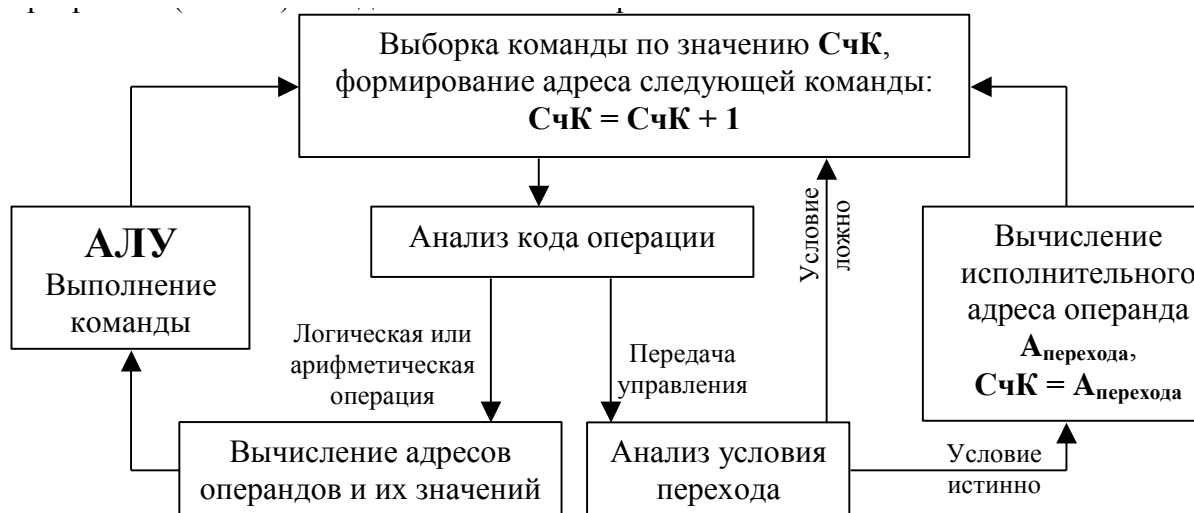


Рис. 27. Алгоритм выполнения процессором программы (рабочий цикл ЦП).

Последовательность действий, происходящая в процессоре при выполнении программы, называется **рабочим циклом процессора**. По ходу рассмотрения материала мы будем уточнять рабочий цикл нашего обобщенного модельного компьютера.

кэширования памяти — размещения части данных в более высокоскоростном запоминающем устройстве. Таким средством является **КЭШ-память** (cache memory) — высокоскоростное устройство хранения данных, используемое для буферизации работы процессора с оперативной памятью. В общем случае, кэш представляет собою аппаратную «емкость», в которой аккумулируются наиболее часто используемые данные из оперативной памяти. Скорость доступа к информации, размещённой в КЭШе, соизмерима со скоростью обработки информации в ЦП. Обмен данными при выполнении программы оперативной памяти. Размещение и команд, и данных в одном КЭШе может приводить к тому, что команды и данные начинают вытеснять друг друга, увеличивая при этом обращения к оперативной памяти. Для исключения недетерминированной конкуренции в КЭШе между командами программы и обрабатываемыми данными современные компьютеры имеют два независимых КЭШа: **КЭШ данных** и **КЭШ команд**, каждый из которых работает со своим потоком информации — потоком команд и потоком операндов.

Общая схема работы КЭШа следующая (Рис. 28).

1. Условно, вся память разделяется на блоки одинакового размера. Обмен данными между КЭШем и оперативной памятью осуществляется **блоками** фиксированного объёма. Здесь мы можем видеть возможное проявление преимущества использования памяти с расслоением, так как загрузка блока из оперативной памяти в КЭШ осуществляется с использованием параллелизма работы «расслоенной» оперативной памяти.

2. Каждый блок имеет спецификатор доступа — **тэг**, в котором находится служебная информация, характеризующая данный блок. В тэге может содержаться информация о том, какой области ОП соответствует содержимое данного блока, занят или свободен блок, производились изменения в данном блоке или нет. Когда процессору нужно обратиться за командой или за данными в ОП, сначала происходит обращение к КЭШу. По содержимому адресного тэга можно однозначно адресовать содержимое блока.

Анализ тэгов блоков КЭШа производится аппаратно. Таким образом, после вычисления исполнительного адреса операнда или команды устройство управления может определить, находится ли соответствующая информация в одном из блоков КЭШ-памяти или нет. Факт нахождения искомых данных в КЭШе называется **попаданием** (hit) – в этом случае данные берутся из КЭШа, и обращение в ОП не осуществляется. Если искомых данных нет в КЭШе, то фиксируется **промах** (cache miss).

3. При возникновении промаха происходит **вытеснение** – обновление содержимого КЭШа. Для этого выбирается блок-претендент на вытеснение, т.е. блок, содержимое которого будет заменено. Стратегия этого выбора зависит от конкретной организации процессора. Существуют КЭШи, вытеснение блоков которых осуществляется *случайным образом*, т.е. номер блока, который должен быть вытеснен, определяется с использованием встроенного генератора случайных чисел. Альтернативой случайного вытеснения является *вытеснение наименее «популярного» блока КЭШа*, т.е. блока, к содержимому которого происходило наименьшее число обращений (LRU — Least-Recently Used).

4. Отдельно следует обратить внимание на организацию вытеснения блока в КЭШе данных, т.к. содержимое блоков КЭШа может не соответствовать содержимому памяти (это возникает при обработке команд записи данных в память). В этом случае также возможно использование нескольких стратегий вытеснения. Первая — **сквозное кэширование** (write-through caching): при выполнении команды записи данных обновление происходит как в КЭШе, так и в оперативной памяти. Таким образом, при вытеснении блока из КЭШа происходит только загрузка содержимого нового блока. Данная стратегия оправдана, т.к. статистические исследования показывают, что частота чтения данных превосходит частоту их записи на порядок. Другой стратегией является **кэширование с обратной связью** (write-back caching), суть которой заключается в использовании специального *тега модификации* (dirty bit). При выполнении команды записи по адресу, содержимое которого кэшируется в одном из блоков, происходит обновление соответствующей этому адресу информации только в блоке КЭШа, а также установка в блоке тега модификации. Соответственно, при вытеснении блока осуществляется контроль за содержимым тега. Если тег модификации установлен, то содержимое блока перед вытеснением «сбрасывается» в память. Тем самым минимизируется частота выполнения операции записи в память.

Использование КЭШ-памяти позволяет получить следующие *преимущества*. Во-первых, сокращается количество обращений к ОЗУ – обращений, как по выборке команд, так и по выборке операндов. Во-вторых, существенно увеличивается скорость доступа к памяти в случае использования ОЗУ с «расслоением», так как обмены блоков с памятью будут проходить практически параллельно (когда мы работаем с группой подряд идущих слов).

Естественно, при использовании КЭШ-памяти возникают и некоторые *проблемы*. Во-первых, усложнение логики процессора. Организация и использование КЭШ-памяти в процессоре развивает рабочий цикл процессора модельного компьютера, рассмотренный выше: при выборке очередных команд, получении операндов команд и записи результатов выполнения команд в ОЗУ добавляются схемы организации использования КЭШ-памяти. Во-вторых, если КЭШ один (т.е. потоки команд и данных приходятся на один КЭШ), то один из потоков может начать «довлеть» над другим, так как характеристики потоков команд и данных разные (поток команд обладает свойством локализации, поток данных этим свойством не обладает). Поэтому получили распространение архитектуры, в которых КЭШ разделяется на КЭШ команд и КЭШ данных. Это также позволяет повысить производительность системы.

Прерыванием называется событие в компьютере, при возникновении которого в системе предусмотрена предопределенная последовательность действий, включающая стандартную реакцию процессора на прерывание и этап программной обработки прерываний (функция ОС). Состав прерываний — множество разновидностей событий, на возникновение которых предусмотрена стандартная реакция центрального процессора, — фиксирован и определяется конструктивно при разработке компьютера. **Аппарат прерываний** компьютера позволяет организовывать стандартную обработку прерываний, возникающих при функционировании вычислительной системы. Традиционно прерывания разделяются на два типа: **внутренние прерывания** и **внешние прерывания**.

Внутренние прерывания инициируются схемами контроля работы процессора.

Внешние прерывания — события, возникающие в компьютере в результате взаимодействия центрального процессора с внешними устройствами. Примером внешнего прерывания может служить событие, связанное с вводом символа с клавиатуры персонального компьютера.

Обработка прерывания предполагает две стадии: **аппаратную**, которая включает предопределённую реакцию процессора на возникновение прерывания, и **программную**, которая предполагает выполнение специальной программы обработки прерывания, являющейся частью операционной системы



Рис. 33. Программный этап обработки прерываний.

1.2.4 Внешние устройства

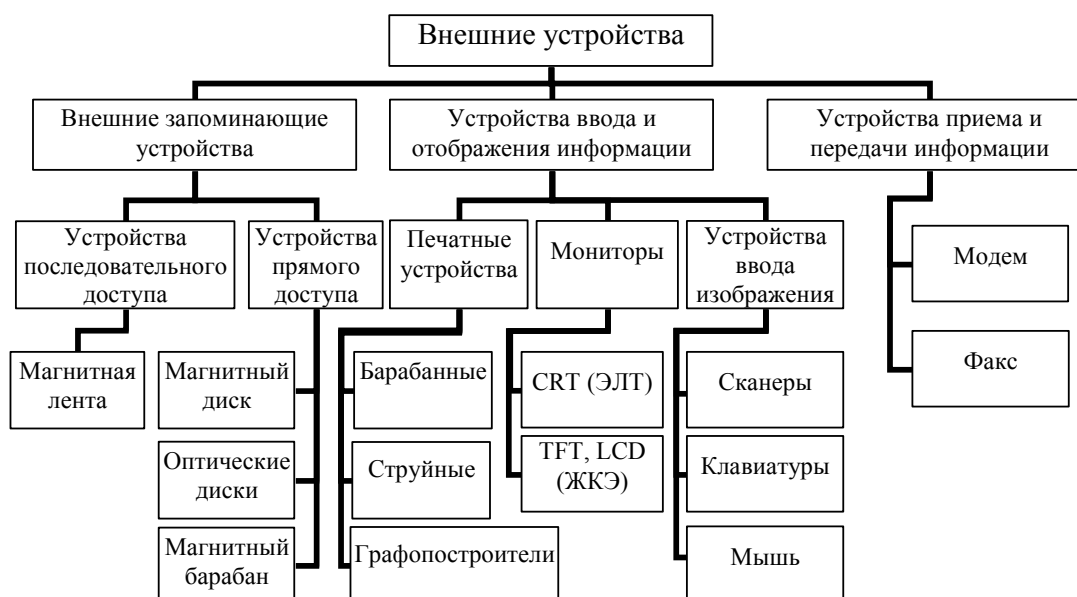


Рис. 34. Иерархия внешних устройств.

Мы более подробно остановимся на характеристиках и особенностях использования внешних запоминающих устройств, как наиболее интенсивно используемых программами внешних устройств вычислительных систем.

1.2.4.1 Внешние запоминающие устройства

Внешние запоминающие устройства (ВЗУ) предназначены для организации хранения данных и программ. Обычно обмен с ВЗУ происходит некоторыми порциями данных, которые называются *записями*. Данные, размещенные на ВЗУ, представляются в виде последовательности записей. Существует категория ВЗУ, называемые **блочными устройствами**, которые допускают выполнение обменов исключительно записями фиксированного размера — *блоками*. Примером блочных устройств могут служить различные типы магнитных дисков. Обычно размер блоков (**физических блоков**), обмен которыми может осуществляться с блочными устройствами, определяется аппаратно и может зависеть от конкретной модели и типа устройства. Альтернативой блочным ВЗУ являются устройства, аппаратно допускающие обмен записями произвольного размера. Примером таких устройств являются устройства хранения информации на магнитных лентах.

ВЗУ могут разделяться на две группы по *возможностям доступа к хранящимся данным*. Первая группа — устройства, аппаратно допускающие как операции чтения, так и операции записи. Примером устройств данной группы может служить жесткий диск. Вторая группа — устройства, позволяющие выполнять только операции чтения данных, например, в эту группу входят устройства CD-ROM (compact disk read-only memory), DVD-ROM (digital video/versatile disc read-only memory).

Внешние запоминающие устройства могут, также подразделяться на **устройства прямого доступа** и **устройства последовательного доступа**. Рассмотрим принципы организации и общие характеристики устройств, принадлежащих каждой из этих групп.

Устройства последовательного доступа — это устройства, при доступе к содержимому произвольной записи которых «просматриваются» все записи, предшествующие искомой. Рассмотрим в качестве примера ВЗУ последовательного доступа устройство хранения данных на магнитной ленте. На магнитной ленте каждая

запись имеет специальные маркеры начала и конца. Также, на каждой ленте размещаются маркеры начала и конца ленты (Рис. 35).

Рис. 35. Магнитная лента.

Каждая запись на ленте имеет свой логический номер. При возникновении запроса на чтение записи с номером i выполняется следующая последовательность действий:

- устройство перематывает ленту до маркера начала ленты;
- осуществляется последовательный поиск маркеров начала записей, после нахождения i -го маркера считается, что устройство «вышло» на начало искомой записи;
- происходит чтение i -ой записи.

Устройство прямого доступа обеспечивает выполнение операций чтения/записи без считывания дополнительной (предыдущей) информации. Примером устройств прямого доступа могут служить **магнитные диски**, или **дисковые устройства**.

Рассмотрим пример выполнения операции обмена данными, размещенными в одном из секторов. Для задания координат конкретного сектора в устройство управления магнитным диском должны быть переданы:

- номер цилиндра, в котором расположен данный сектор, — N_c ;
- номер дорожки, на которой размещается сектор, — N_t ;
- номер сектора — N_s .

После получения координат сектора (N_c , N_t , N_s) выполняется следующая последовательность действий:

- шаговый двигатель перемещает блок головок в цилиндр N_c ;
- включается головка чтения/записи, соответствующая номеру дорожки N_t ;
- как только головка чтения/записи позиционируется над началом искомого сектора N_s , запускается выполнение операции чтения (или записи).

Таким образом, мы видим, что для выполнения операций обмена с магнитным диском не производится чтение какой-либо дополнительной информации с диска, т.е. обеспечивается «прямой доступ» к информации.

Производительность внешнего запоминающего устройства — время доступа к хранящейся информации — во многом определяется наличием и продолжительностью механических операций, которые необходимо провести при обмене. Так, время обмена с

Следующее устройство, которое мы рассмотрим, — это **магнитный барабан** (Рис. 37). В данном приборе также имеется электродвигатель, к оси которого прикреплен массивный барабан, поверхность которого покрыта электромагнитным слоем. Двигатель раскручивает барабан до достаточно высокой постоянной скорости. Помимо этого имеется фиксированная штанга, на которой расположены головки чтения-записи. Под каждой головкой логически можно выделить дорожку, которая называется **треком**. Так же, как и в диске, все дорожки разделены на сектора. Для адресации блока данных в этом случае используется только номер дорожки ($N_{\text{трека}}$) и номер сектора ($N_{\text{сектора}}$). Для того чтобы произвести операцию чтения или записи, устройство управления должно включить головку, соответствующую указанному номеру дорожки; а после этого происходит

ожидание механического поворота цилиндра до выхода головки на начало искомого сектора. Таким образом, по сравнению с жесткими дисками, в этом устройстве отсутствует механическая составляющая выхода головки на нужный трек, поэтому данный тип устройств считается более высокоскоростным.

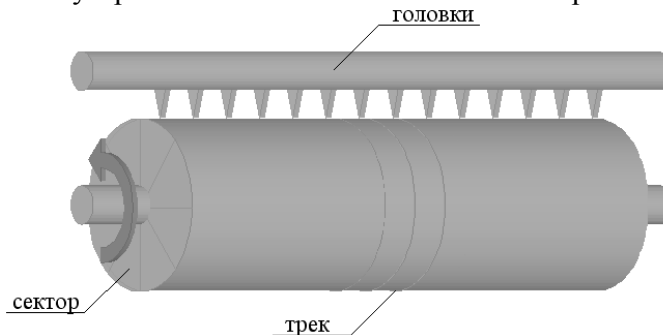


Рис. 37. Принцип устройства магнитного барабана.

И, наконец, отметим т.н. **память на магнитных носителях (доменах)** – магнитно-электронное ВЗУ прямого доступа. Под **доменом** понимается некоторая элементарная единица, способная сохранять свою намагниченность в течение длительного промежутка времени. Домен может быть намагничен одним из двух способов (отмеченные на Рис. 38 либо как «плюс-минус», либо как «минус-плюс»).

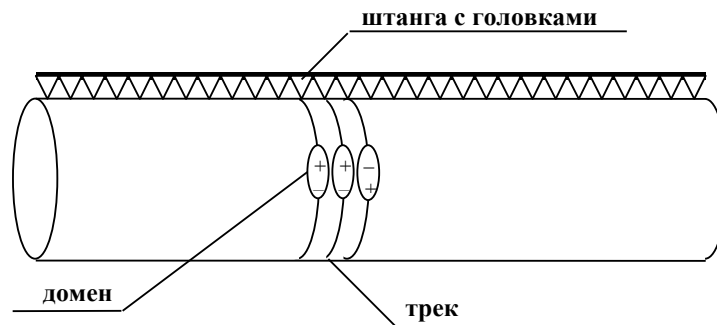


Рис. 38. Принцип устройства памяти на магнитных доменах.

Принцип работы устройства памяти на магнитных доменах основан на том, что под воздействием магнитно-электронных эффектов магнитные домены разгоняются вдоль своего трека до некоторой постоянной скорости. В остальном же принцип работы данного класса устройств ничем не отличается от работы магнитных барабанов. Соответственно, из-за того, что в данном устройстве нет механической составляющей, оно является еще более высокоскоростным по сравнению с предыдущими устройствами.

1.2.4.2 Модели синхронизации при обмене с внешними устройствами

Синхронная работа с ВУ. При синхронной организации обмена в момент обращения к внешнему устройству программа будет приостановлена до момента завершения обмена (Рис. 39). Тем самым при использовании такой модели в системе возникали задержки, которые снижали эффективность функционирования ВС.

Асинхронная работа с ВУ. При асинхронной организации работы внешних устройств последовательность событий, происходящих в системе, следующая:

1. Для простоты изложения будем считать, что в системе прерываний компьютера имеется специальное внутреннее прерывание «обращение к системе», которое инициируется выполнением программой специальной команды. Программа инициирует прерывание «обращение к системе» и передает заказ на выполнение обмена, параметры заказа могут быть переданы через специальные регистры, стек и т.п. В операционной системе происходит обработка прерывания, при этом конкретному драйверу устройства передается заказ на выполнение обмена.

2. После завершения обработки «обращения к системе» программа может продолжить свое выполнение, или может быть запущено выполнение другой программы.

3. По завершении выполнения обмена происходит прерывание, после обработки которого программа, выполнявшая обмен, может продолжить свое выполнение.

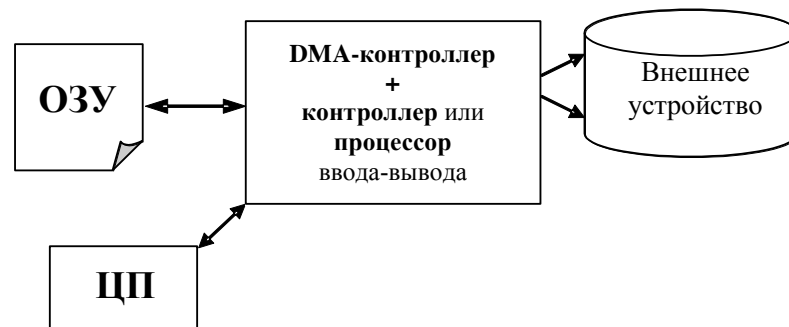
1.2.4.3 Поток данных. Организация управления внешними устройствами

Простейшей моделью является **непосредственное управление процессором** внешними устройствами (Рис. 40). Это означает, что центральный процессор фактически «интегрирован» со схемами управления внешними устройствами, имеет специальные команды управления ими, а также путем интерпретации последовательности команд управления осуществляет управление обменом. Т.е. процессор подает команды устройству на перемещение головок обмена, на включение той или иной головки, на ожидание и синхронизации прихода содержательной информации и пр. Помимо указанного потока команд, центральный процессор обрабатывает и поток данных: он считывает информацию, участвующую в обмене, со специальных регистров и переносит ее в оперативную память (либо же производит обратные манипуляции). Таким образом, и поток управления, и поток данных проходит через центральный процессор, что само по себе является трудоемкой задачей, к тому же эта модель подразумевает лишь синхронную реализацию.

Следующая модель предлагает **синхронное управление** внешними устройствами с использованием контроллеров внешних устройств (Рис. 41). Данная модель появилась вслед за появлением внешних устройств, для которых имелись электронные схемы управления этими устройствами — **контроллеры**, — взявшие на себя часть работ центрального процессора по управлению обемами. В этом случае контроллер взаимодействует с центральным процессором блоками больших размеров, при этом контроллер может самостоятельно выполнять некоторые работы по непосредственному управлению ВЗУ (например, пытаться локализовать и исправить возможные ошибки, которые могут случиться при чтении или записи данных). Но исторически такой тип управления ВЗУ изначально был синхронным: процессор посылает устройству команды на обмен и ожидает, когда этот обмен завершится. Что касается потока данных, то ничего нового в данной модели не представлено: процессор по-прежнему считывает их со специальных регистров внешнего устройства и помещает их в оперативную память.

Вслед за предыдущим типом устройств появились устройства, позволяющие осуществлять **асинхронное управление** с использованием контроллеров ВЗУ (Рис. 41). В этом случае центральный процессор подает команду на обмен и не дожидается, когда эту команду отработают контроллер и устройство, т.е. процессор может продолжить обработку каких-то задач. Но для осуществления указанной модели необходимо, чтобы в системе был реализован аппарат прерываний.

Затем исторически появились т.н. **контроллеры прямого доступа к памяти** (DMA — Direct Memory Access, Рис. 42). Контроллеры данного типа исключили центральный процессор из обработки потока данных, взяв эту функцию на себя. В данной модели предполагается, что центральный процессор занимается лишь обработкой потоком управляющей информации, а данные перемещаются между ВЗУ и ОЗУ уже без его участия.



И, наконец, последняя модель основана на использовании **процессора** или **канала ввода-вывода** (Рис. 42). В этом случае предполагается наличие специализированного компьютера, который имеет свой процессорный элемент, свою оперативную память, который функционирует под управлением своей ОС, и этот компьютер располагается логически между центральным процессором и внешними устройствами. В функции подобных процессоров или каналов входит высокоуровневое управление внешних устройств. В этом случае центральный процессор оперирует с внешними устройствами в форме высокоуровневых заказов на обмен. Соответственно, реализация непосредственного управления конкретным ВЗУ осуществляется в процессоре ввода-вывода (в частности, в нем может происходить многоуровневая фиксация ошибок, он может осуществлять аппаратное кэширование обменов к конкретному устройству и пр.).

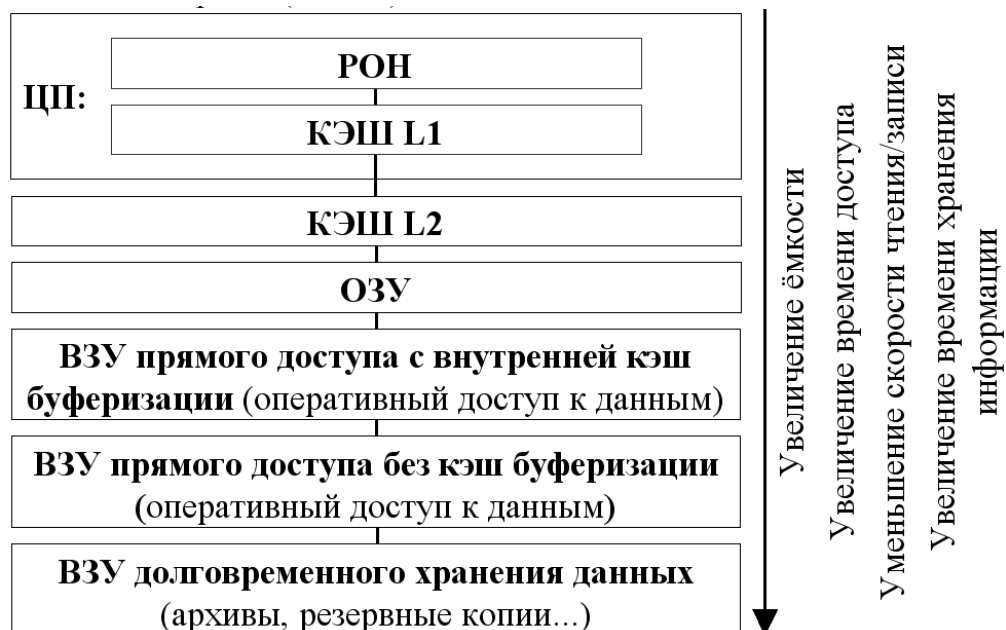


Рис. 43. Иерархия устройств хранения информации.

19. Операционные системы, основные функции. Типы операционных систем.

Операционная система — это комплекс программ, в функции которого входит обеспечение контроля за существованием, использованием и распределением ресурсов вычислительной системы. Напомним, что вычислительная система может включать в свой состав как физические, так и виртуальные ресурсы. Чтобы дать более ясную картину того, что же мы будем считать операционной системой, разберем детально её функции.

Физические ресурсы (устройства) — компоненты аппаратуры компьютера, используемые на программных уровнях ВС или оказывающие влияние на функционирование всей ВС. Совокупность физических ресурсов составляет аппаратный уровень вычислительной системы.

Логические, или виртуальные, ресурсы (устройства) ВС — устройство/ресурс, некоторые эксплуатационные характеристики которого (возможно все) реализованы программным образом.

Начнем с того, что операционная система обеспечивает **контроль за существованием ресурсов**. Под этим понимается обеспечение операционной системой реализации виртуальных ресурсов и предоставление средств доступа к физическим ресурсам. Для любого ресурса степень его доступности зависит от операционной системы. Существуют ресурсы, которые полностью зависят от того, имеется ли их реализация в операционной системе или нет, если есть, то какая именно это реализация. Примером подобного ресурса служит файловая система: этого ресурса может и не быть в операционной системе, может существовать одна модель, или другая модель, или сразу несколько моделей.

Под **файловой системой (ФС)** будем понимать часть операционной системы, представляющую собой совокупность организованных наборов данных, хранящихся на внешних запоминающих устройствах, и программных средств, гарантирующих именованный доступ к этим данным и их защиту.

Файловая система - это компонент операционной системы, обеспечивающий организацию создания, хранения и доступа к именованным наборам данных - файлам.

Следующий пункт — **использование ресурсов**. Здесь имеется в виду, что операционная система предоставляет все средства, обеспечивающие доступность ресурсов ВС пользователю (точнее программам). При использовании любых ресурсов ВС может возникнуть конкуренция.

И, наконец, **распределение**: под этим будем понимать выбор стратегии распределения и обеспечение всевозможных моделей регламентации доступа.

Любая операционная система опирается на набор базовых сущностей, на основе характеристик которых выстраиваются почти все эксплуатационные свойства конкретной операционной системы. При этом, для различных операционных систем наборы базовых сущностей зачастую различаются: одни основаны на понятии устройства, другие — на понятии файла, третьи — на понятии набора данных. Но в большинстве случаев в состав базовых включается сущность, обозначающая исполняемую программу, задачу, задание или **процесс**. Эта сущность определяет некоторый процесс исполнения последовательности команд, причем здесь может участвовать единственная ветвь

вычислений, а может сразу и несколько параллельных ветвей. Из множества трактовок этой сущности мы выберем понимание ее именно как *процесса*.

Процесс — это совокупность машинных команд и данных, обрабатываемая в рамках вычислительной системы и обладающая правами на владение некоторым набором ресурсов ВС.

Разберемся в этом определении. Понятие **совокупности машинных команд и данных** обозначает то, что принято называть исполняемой программой (т.е. это код и операнды, используемые в этом коде). Далее, под термином **обработки в рамках ВС** будем понимать, что эта программа сформирована и находится в системе в режиме обработки (это может быть и ожидание, и исполнение на процессоре, и т.п.). И, третье, понятие **обладания правами на владение некоторым набором ресурсов** обозначает, по сути, возможность доступа. Отметим, что здесь речь не идет об эксклюзивных правах, поскольку в общем случае это было бы некорректно. Итак, иными словами, процесс можно определить как исполняемую программу, которая введена в систему для ее обработки и с которой ассоциированы некоторые ресурсы вычислительной системы.

Ресурсы, выделяемые процессам, могут быть *двух типов*. Первая категория ресурсов состоит из тех ресурсов, которые выделяются процессу **на эксклюзивных правах**. Это означает, что этот ресурс, пока процесс им владеет, принадлежит ему и только ему, и никакой иной процесс не имеет право работать с данным ресурсом. Вторая категория — это те ресурсы, которые одновременно могут принадлежать двум и более процессам, — такие ресурсы принято называть **разделяемыми ресурсами**. Здесь сделаем небольшое пояснение: то, что разделяемый ресурс может *одновременно принадлежать* нескольким процессам, *не означает*, что к нему возможен *одновременный доступ*. Обозначенная проблема решается на другом уровне посредством использования разных схем синхронизации доступа к разделяемому ресурсу, и об этом речь пойдет несколько позже.

С точки зрения выделения ресурса процессу используются *две стратегии организации* этого выделения. Первый способ — это *предварительная декларация ресурсов*. В этом случае до ввода программы в систему и формирования для нее процесса декларируется перечень тех ресурсов, которыми процесс будет обладать. Например, это может быть перечень областей оперативной памяти, которые будут доступны данному процессу (если система поддерживает механизм виртуальной памяти, то это будет перечень областей виртуальной памяти, доступных процессу). Или же это может быть предельное время центрального процессора, которое может быть потрачено на исполнение данного процесса. Так или иначе, при вводе программы и формировании процесса операционная система постарается выделить *все* необходимые ресурсы, которые были предварительно декларированы. Если в системе нет заказанного ресурса, то она, скорее всего, не станет запускать процесс, который запросил этот ресурс.

Вторая модель — это *динамическое пополнение списка ресурсов*. Данная модель предполагает выделение процессу ресурса уже во время выполнения этого процесса. Это означает, что в системе происходит запуск процесса с выделением ему минимально необходимой области виртуальной памяти, а затем, когда процесс обращается к системе за выделением дополнительной области, то ОС обрабатывает эти запросы соответствующим образом. Отметим также, что на практике также применяются и комбинированные подходы, но во многих системах, с которыми мы сталкиваемся в нашей повседневной жизни, ориентация сделана на динамическую модель выделения ресурсов.

Многие операционные системы разрабатывались и разрабатываются таким образом, чтобы обладать следующими важными *свойствами*: надежность, защита, эффективность и предсказуемость.

Надежность означает, что система должна быть надежной как программный комплекс, т.е. число программных ошибок в системе должно быть сведено к минимуму и должно быть соизмеримо с количеством возможных аппаратных сбоев.

Защита информации на сегодняшний день является одним из основных требований, предъявляемых к системе. ОС должна обеспечивать защиту информации и ресурсов от несанкционированного доступа. Эта проблема на сегодняшний день остаётся открытой.

Свойство **эффективности** означает, что функционирование системы должно удовлетворять некоторым требованиям – критериям эффективности, которые, по сути, являются оценкой соответствия.

И, наконец, это **предсказуемость** системы, являющаяся также одним из важных свойств ОС, поскольку большинство систем, которые, так или иначе, являются массово распространенными, при возникновении разного рода форс-мажорных обстоятельств должны вести себя строго определенным способом. Это свойство должно очерчивать круг всевозможных проблем, которые могут возникнуть в той или иной ситуации, предопределять последствия этих проблем, а также подразумевать устойчивость системы к возникновению сбоев (как аппаратных, так и программных).

1.3.1 Структура ОС

Существует множество взглядов, касающихся структуры операционной системы, и в этом разделе речь пойдет о некоторых из них.

Простейшая структурная организация основана на представлении операционной системы в виде композиции следующих компонентов (Рис. 72).

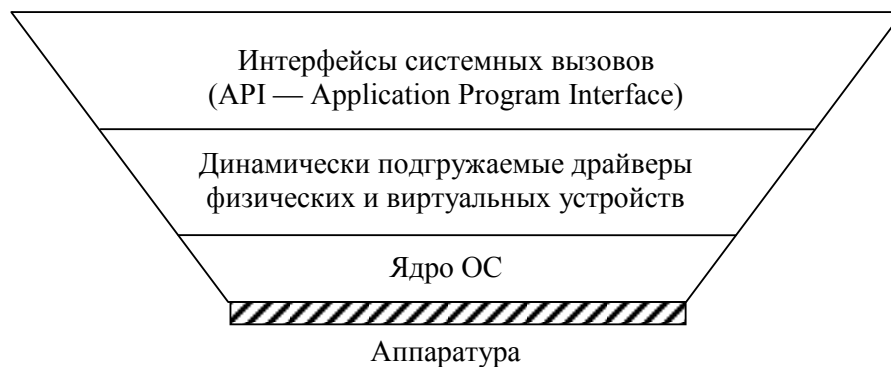


Рис. 72. Структурная организация ОС.

Ядро (kernel) ОС — это резидентная (постоянно размещаемая в ОП) часть ОС, реализующая некоторую базовую функциональность ОС и работающая в режиме супервизора (т.е. в привилегированном режиме). Итак, по определению ядро обеспечивает реализацию некоторого набора функций операционной системы. Это может быть очень большой набор функций, а может быть маленький — все зависит от конкретной реализации системы. Ядро может включать в свой состав драйверы основных физических или виртуальных устройств.

Над уровнем ядра может надстраиваться следующий уровень — это **уровень динамически подгружаемых драйверов физических и виртуальных устройств**. Под **динамически подгружаемыми** понимается то, что в зависимости от ситуации состав этих драйверов *при установке и загрузке* системы может меняться. Соответственно, эти драйверы можно поделить на две категории: **резидентные драйверы** и **нерезидентные**.

Резидентные драйверы подгружаются в систему в процессе ее загрузки и находятся в ней до завершения ее работы. Примером резидентного драйвера может быть драйвер физического диска. **Нерезидентные драйверы** вызываются операционной системой на сеанс работы с соответствующими устройствами (например, драйвер флэш-памяти).

Отметим, что большинство современных операционных систем имеют в своем составе набор драйверов широкого спектра конкретных физических устройств и, в

частности, физических дисков. Поэтому зачастую при смене устройства драйвер менять не надо: он уже есть в системе. Но при этом системе незачем держать драйвера всех устройств в оперативной памяти. Соответственно, следуя той или иной стратегии, будут загружаться драйверы тех физических устройств, которые реально будут обслуживаться системой. Стратегии могут быть различными, одной из них: может быть явное указание системе списка драйверов, которые необходимо подгрузить (в этом случае, если в списке что-то будет указано неправильно, то соответствующее устройство, возможно, просто не будет работать). Вторая стратегия предполагает, что система при загрузке самостоятельно сканирует подключенное к ней оборудование и выбирает те драйверы, которые должны быть подгружены для обслуживания найденного оборудования.

Драйвер физического устройства — программа, основанная на использовании команд управления конкретного физического устройства и предназначенная для организации работы с данным устройством. Драйвер физического устройства скрывает от пользователя детали управления конкретным физическим устройством и предоставляет пользователю упрощенный программный интерфейс работы с устройством.

Драйвер логического/виртуального ресурса — это программа, обеспечивающая существование и использование соответствующего ресурса, для этих целей при его

реализации возможно использование существующих драйверов физических и виртуальных устройств.

В общем случае драйверы могут работать как в привилегированном режиме, так и в пользовательском.

И, наконец, некоторой логической вершиной рассматриваемой структуры ОС будут являться *интерфейсы системных вызовов* (API — Application Program Interface). Под *системным вызовом* будем понимать средство обращения процесса к ядру операционной системы за выполнением той или иной функции (возможности, услуги, сервиса). Примерами системных вызовов являются открытие файла, чтение/запись в него, порождение процесса и т.д. Отличие обращения к системному вызову от обращения к библиотеке программ заключается в том, что библиотечная программа присоединяется к исполняемому коду процесса, поэтому вычисление библиотечных функций будет происходить в рамках процесса. Обращение к системному вызову — это вызов тех команд, которые инициируют обращение к системе. Как уже отмечалось выше, инициацией обращения к операционной системе может служить либо прерывание, либо исполнение специальной команды. Следует понимать различие между системным

вызовом и библиотечной функцией. Например, осуществляя работу с файлом, имеется возможность работы с ним посредством обращения к системным вызовам либо посредством использования библиотеки ввода-вывода. В последнем случае в тело процесса включаются дополнительные функции из данной библиотеки, а уже внутри данных функций происходит обращение к необходимым системным вызовам.

Итак, существует несколько подходов к структурной организации операционных систем. Один из них можно назвать классическим: он использовался в первых операционных системах и используется до сих пор — это подход, основанный на использовании *монолитного ядра*. В этом случае ядро ОС представляет собою единую монолитную программу, в которой отсутствует явная структуризация, хотя, конечно, в ней есть логическая структуризация. Это означает, что монолитное ядро содержит фиксированное число реализованных в нем базовых функций, поэтому модификация функционального набора достаточно затруднительна (необходима практически полная переделка ядра). Устройство монолитного ядра напоминает физическую организацию первых компьютеров: в них также нельзя было выделить отдельные физические функциональные блоки — все было единым, монолитным и интегрированным друг с другом. Аналогичными свойствами обладают одноплатные компьютеры, у которых все необходимые компоненты (ЦПУ, ОЗУ и пр.) расположены на одной плате, и, чтобы что-то изменить в этой конфигурации, требуются соответствующие инженерные знания.

На Рис. 73 проиллюстрирована структурная организация классической системы Unix. В данном случае ядро имеет фиксированный интерфейс системных вызовов. В нем реализовано управление процессами, а также драйвер файловой системы, реализована вся логика системы по организации работы с устройствами (которые можно разделить на байт-ориентированные и блок-ориентированные) и пр.

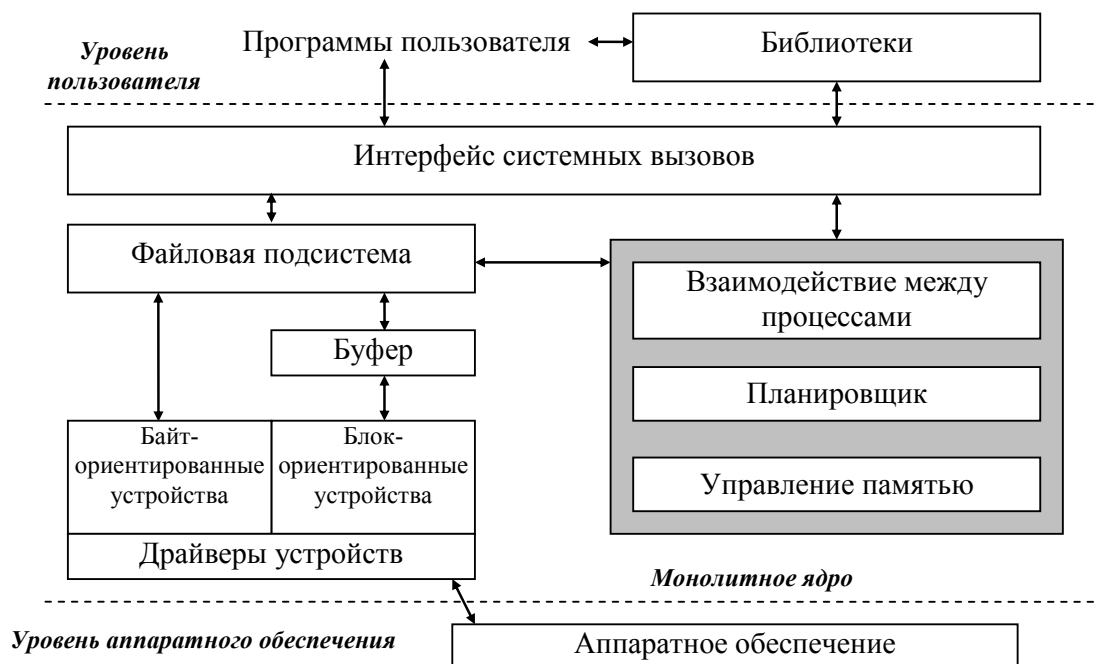


Рис. 73. Структура ОС с монолитным ядром.

Что касается достоинств данного подхода, то можно утверждать, что для конкретного состава функциональности и логики ядра это будет *наиболее эффективное* решение (т.к. оно имеет минимальное количество интерфейсных сочленений, связанных со структуризацией). Однако при таком подходе отсутствует универсальность, и внутренняя организация ядра рассчитана на конкретную реализацию. Недостаток в этом случае заключается в необходимости перепрограммировать ядро при внесении изменений, и это является прерогативой разработчика. Соответственно, для внесения новой функциональности пользователю системы приходится обращаться к разработчику, что зачастую ведет к материальным затратам.

Альтернативу данному подходу предлагают **многослойные операционные системы**. В этом случае все уровни разделяются на некоторые функциональные слои. Здесь можно провести аналогию с моделью сетевых протоколов. Между слоями имеются фиксированные интерфейсы. Управление происходит посредством взаимодействия соседних слоев. Поскольку любая структуризация снижает эффективность (программа, написанная в виде одной большой функции, работает быстрее, чем аналогичная программа, разбитая на подпрограммы, т.к. любое обращение к подпрограмме ведет к накладным расходам), то подобные системы обладают более низкой эффективностью.

Итак, каждый слой предоставляет определенный сервис вышестоящему слою. Деление на слои является индивидуальным для каждой конкретной операционной системы. Это может быть слой файловой системы, слой управления внешними устройствами и т.д. Тогда модернизация подобных систем сводится к модернизации соответствующих слоев. Вследствие чего проблема несколько упрощается, но при этом остаются ограничения на структурную организацию (например, имея слой файловой

системы, можно заменить его другим вариантом этого слоя, но использовать одновременно две различные файловые системы не представляется возможным).

Третий подход предлагает использовать **микроядерную архитектуру** (Рис. 74). Функционирование операционных систем подобного типа основывается на использовании т.н. **микроядра**. В этом случае выделяется *минимальный* набор функций (например, первичная обработка прерываний и некоторые функции управления процессами), которые включаются в ядро. Вся остальная функциональность представляется в виде драйверов, которые подключаются к ядру посредством некоторого стандартного интерфейса.

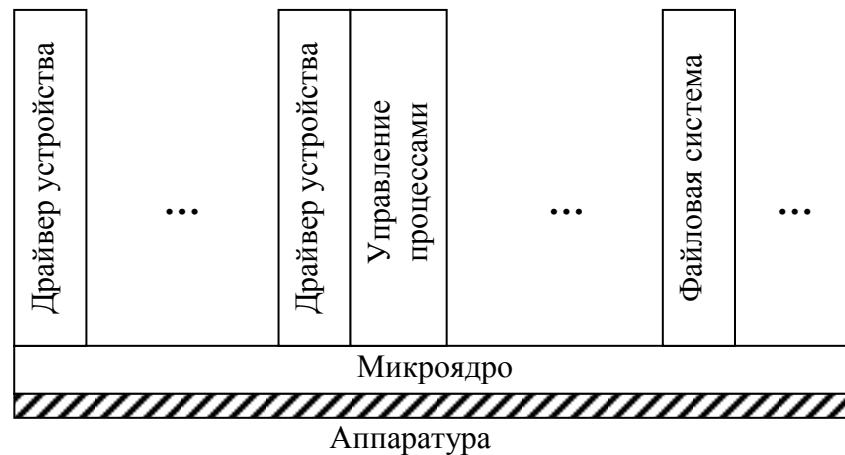


Рис. 74. Структура ОС с микроядерной архитектурой.

Такая архитектура получается хорошо расширяемой, она почти не имеет никаких ограничений по количеству подключаемых драйверов и их функциональному наполнению; требуется только соблюдение драйвером интерфейса для обращения к микроядру. Таким образом, данная архитектура представляется highly технологичной, хорошо подходит для применения в современных многопроцессорных вычислительных системах (например, в SMP-системах, тогда можно распределять драйверы по различным процессорам и получать соответствующую эффективность).

Микроядерная система может служить основой для надстройки над микроядром разных операционных систем. В частности, такой подход используется в ряде систем, в основе которых используется микроядро системы Mach.

Итак, только что были продемонстрированы достоинства данного подхода. Что касается недостатков, то они непосредственно связаны с достоинствами и проявляются в значительном возрастании *накладных расходов*. Положим, процесс обращается к файловой системе, чтобы произвести обмен с конкретным файлом посредством соответствующего системного вызова. Драйвер файловой системы, получив запрос от процесса, перерабатывает его в последовательность запросов на обмен с диском (пусть, сначала это будет виртуальный диск). После чего файловая система обращается к микроядру, которое, в свою очередь, находит драйвер виртуального диска и передает ему соответствующий запрос. Драйвер виртуального диска определяет, с каким физическим диском будет происходить обмен, и трансформирует поступивший ему запрос в запросы к этому физическому диску, которые ему и передаются по той же схеме. Таким образом, один запрос распадается на множество подзапросов, следующих от драйвера через микроядро к другому драйверу, из-за чего эффективность системы снижается.

Напоследок отметим, что в реальности используются системы, получаемые комбинацией указанных подходов.

1.3.2 Логические функции ОС

Функциональность ОС можно представить в виде объединения некоторого фиксированного количества блоков функций. Состав этого набора варьирует от системы к системе, но в большинстве случаев можно выделить следующие функции: *управление процессами*, *управление оперативной памятью*, *планирование* и, наконец, *управление данными, файловой системой и устройствами*, а также в последнее время стали добавлять блок функциональности *сетевого взаимодействия*.

Функция **управления процессами** охватывает проблемы формирования процессов, поддержания жизненного цикла процесса, проблемы организации взаимодействия процессов (т.е. организацию взаимодействия процесса с системой в целом и с другими процессами в частности) и работы процессов с ресурсами. Функция управления процессами является ключевой для всех операционных систем.

Функция **управления оперативной памятью** во многом определяется той аппаратурой, на которой реализуется ОС. С точки зрения программной составляющей, управление ОП включает компоненты, связанные с выбором стратегии организации ОП (в частности, реализуется поддержка аппарата виртуальной памяти), решает задачи защиты ОП от несанкционированного доступа, задачи корректности работы процесса с выделенной ему ОП. Некоторые из задач управления ОП пересекаются с функциями планирования (например, задачи выделения и изъятия памяти у процессов).

Функция **планирования** во многом определяет базовые характеристики ОС. Ранее говорилось, что в системе практически всегда возникает ситуация конкуренции за обладание теми или иными ресурсами. При этом сразу же возникает задача регламентации доступа конкурирующих процессов к ресурсам (т.е. задача обработки очередей запросов к ресурсам). Классическая задача планирования – *распределение времени центрального процессора* (т.е. планирование доступа процессов к центральному процессору). Ещё одна задача планирования – *организация и обработка очередей обмена*. В процессе функционирования системы формируется поток запросов на обмен, и очень часто этот поток может превышать пропускную способность устройства, то есть образуется конкуренция по доступу к устройству — выстраивается очередь запросов на обмен. Возникает проблема планирования очередей обмена. При этом необходимо учитывать приоритетность при обработке категорий запросов, которые идут от ОС и которые идут от пользователей. Следующая задача планирования – *обработка прерываний*. При обработке прерываний также есть приоритетность, так как прерывание иногда требует достаточно больших ресурсов системы и может возникать очередь прерываний.

Следующая функция ОС – **управление устройствами и файловой системой**. Здесь рассматривается управление внешними устройствами, и файловая система рассматривается как виртуальное устройство, которое является достаточно значимым для операционной системы. Эту функцию можно разделить на управление устройствами, которые не требуют оперативного доступа, и управление устройствами, которые требуют оперативный доступ (которые могут содержать важную информацию и которые должны обеспечивать надёжность сохранения этой информации). Организация управления внешними устройствами должна осуществляться надёжно и эффективно. Это достигается посредством многоуровневой буферизации запросов к внешним устройствам, использованием достаточно сложных схем обеспечения надёжности обменов с внешними устройствами (например, программные средства дублирования) и т.д.

Следующая функция – **обеспечение сетевого взаимодействия**. Практически любая современная ОС должна иметь средства взаимодействия с другими компьютерами в сети, то есть ОС должна обеспечивать функционирование и реализацию сетевых протоколов. При этом заметим, что не обязательно весь реализованный в системе стек протоколов должен являться компонентом ОС.

Следующая очень важная функция – функция **обеспечения безопасности**. История возникновения проблемы компьютерной безопасности связана с появлением многопользовательских вычислительных систем, а именно, прежде всего, необходимо было обеспечить, чтобы один зарегистрированный в системе не мог добраться до информации другого зарегистрированного в системе пользователя и чтобы незарегистрированный пользователь не мог получить доступ к системе. Развитие сетевого взаимодействия существенно усложнило проблему безопасности, так как из-за концепции открытых интерфейсов (ISO/OSI, TCP/IP) у компьютера появилось множество логических «входов», через которые программы других компьютеров могут связываться с данным. Появилось понятие компьютерной атаки на сетевое устройство с целью нарушения защиты (либо с целью несанкционированного доступа к данным, либо с целью нарушения функциональности устройства). То есть функция безопасности должна обеспечивать, во-первых, устойчивость системы к возможным атакам, а, во-вторых, анализ функционирования системы и выявление попыток вторжения в систему (это также важно, поскольку очень редко, когда взлом происходит сразу). Это очень сложная проблема, которая до настоящего момента является открытой. Следующий аспект обеспечения безопасности – это минимизация ущерба в случае вторжения.

1.3.3 Типы операционных систем

Операционные системы можно классифицировать с точки зрения критериев эффективности и стратегий использования центрального процессора. Можно выделить три основных класса операционных систем: **пакетные** операционные системы, системы **разделения времени** и системы **реального времени**. Остановимся на каждой из них поподробнее.

Пакетная операционная система — это система, критерием эффективности функционирования которой является максимальная загрузка центрального процессора (т.е. минимизация накладных расходов). Иными словами, отношение всего времени работы процессора ко времени исполнения пользовательских программ должно быть близко к единице. Традиционно пакетные системы предназначались для решения расчетных задач, т.е. задач, требующих определенного объема времени работы процессора. Как следует из названия, эти системы оперируют термином **пакет программ**.

Пакет программ — это некоторая совокупность программ, которые системе необходимо обработать. Особенность пакетных систем прослеживается в стратегии переключения выполнения процессов на процессоре – переключение выполнения процессов происходит только по одной из трех причин.

Первая причина — завершение выполнения процесса (в силу успешного перехода на точку завершения программы или же в силу возникновения ошибки).

Вторая причина — обращение к внешнему устройству с целью осуществления обмена, т.е. возникновение прерывания по вводу-выводу, поскольку операция обмена так или иначе требует какого-то минимального интервала времени.

И, наконец, третья причина — фиксация факта заикливания процесса. В принципе точно определить факт заикливания программы сложно, но все-таки возможно. На практике зачастую под фактом заикливания считают исчерпание процессорного времени (положим, полтора часа).

Очевидно, что переключение процессов в подобных системах происходит лишь по необходимости, а это означает, что происходит редкое обращение к функции ОС смены контекстов обрабатываемых процессов, что ведет к максимальному снижению накладных расходов. В подобных системах степень полезной загрузки процессора составляет от 90% и выше.

Следующая модель — **система разделения времени**. Данная модель может рассматриваться как развитие модели пакетных систем. В дополнение ко всем свойствам пакетных систем необходимо добавить дополнительную характеристику – для каждого

процесса в системе определяется **квант** процессорного времени, который может быть одновременно использован процессом. Под **квантом** времени центрального процессора понимается некоторый фиксированный операционной системой промежуток времени работы процессора. Соответственно, переключение процессов происходит по тем же причинам, что и в пакетных системах (завершение процесса, возникновение прерывания, фиксация факта заикливания), но необходимо добавить еще одну причину — исчерпанся выделенный квант времени.

Критерием эффективности подобных систем служит вовсе не загрузка процессора, а минимизация времени отклика системы на запрос пользователя (положим, если пользователь набирает текст в текстовом редакторе, то будет важно, чтобы набранные им только что символы отображались на экране достаточно быстро, иначе работать с системой ему будет неудобно). Очевидно, что в подобных системах происходит частая смена контекстов, что связано с большими накладными расходами. В подобных системах эффективность может составлять порядка 30–40%, а, соответственно, 60–70% будут составлять накладные расходы.

Варьируя размерами кванта времени, можно получать системы для решения тех или иных задач. Увеличивая квант времени до некоторого среднего размера (порядка нескольких секунд), можно получить пакетную систему, ориентированную на обработку отладочных программ. А если увеличить размер кванта до бесконечности, получится пакетная система в чистом виде.

При организации планирования времени центрального процессора в системах разделения времени необходимо следующее. Во-первых, разделить все процессы на группы по некоторым критериям (например, интерактивные, отладочные и т.д.). Во-вторых, для каждой из этих групп определить квант времени ЦП, который будет выделяться процессу из конкретной группы. В-третьих, решить вопрос справедливой организации вычислительного процесса, т.е. определить приоритеты для каждой из категорий процессов. При этом в критерии смены обрабатываемого процесса можно добавить ещё один пункт – появление процесса из более приоритетной группы. В-четвёртых, приоритеты категорий процессов должны определяться по расписанию (например, в зависимости от времени суток).

Еще один класс систем представляют **операционные системы реального времени**. Это специализированные системы, которые предназначены для функционирования в рамках вычислительных систем, обеспечивающих управление и взаимодействие с различными технологическими процессами. При разработке подобных систем все функции планирования ориентированы на обработку некоторого фиксированного набора событий, при возникновении любого из которых гарантируется обработка этого события за некоторый промежуток времени, не превосходящий определенного предельного значения.

Сфер применения систем реального времени в жизни очень много. Достаточно часто системы реального времени строятся под конкретные задачи. Выделяют различные группы систем реального времени: жесткого времени (например, управление бортовой системой самолёта), мягкого времени и пр.; но основной принцип их функционирования одинаков и подобен тому, который был проиллюстрирован выше.

И, в заключение, кратко остановимся на рассмотрении **сетевых и распределенных операционных систем**. Как уже отмечалось выше, одиночные однопроцессорные системы уходят в прошлое, и во многих случаях процессорный элемент или компьютерный элемент рассматривается как составляющая многопроцессорных или многомашинных ассоциаций. И с этой точки зрения операционные системы можно разделить на две категории.

В первую категорию можно отнести т.н. *сетевые* ОС (Рис. 75). Сетевая операционная система — это система, обеспечивающая функционирование и взаимодействие вычислительной системы в пределах сети. Это означает, что сетевая ОС устанавливается на каждом компьютере сети и обеспечивает функционирование распределенных приложений, т.е. тех приложений, реализация функций которых распределена по разным компьютерам сети. Примеров можно привести достаточно много. Так, почтовое приложение может быть распределенным: есть функции перемещения, есть сервер-получатель, есть клиентская часть, обеспечивающая интерфейс работы пользователя с указанным сервером.



Рис. 75. Структура сетевой ОС.

Вторую категорию составляют *распределенные* ОС (Рис. 76). Распределенной операционной системой считается система, функционирующая на многопроцессорном или многомашинном комплексе, в котором на каждом из узлов функционирует отдельное ядро, а сама система обеспечивает реализацию распределенных возможностей ОС (т.н. сервисы или услуги). Примером распределенных функций может служить функция управления заданиями (напомним, что в кластерных системах задание может представлять собою целое множество процессов, и ставится задача распределить эти процессы по имеющимся процессорным узлам). Другим примером может служить распределенная файловая система. Традиционная файловая система ОС Unix просто не справится с потоками информации между узлами многопроцессорных систем, поэтому необходимы принципиально новые решения организации хранения и доступа к файлам.

Обработка прерываний.

В каждой вычислительной машине имеется предопределенный, заданный при разработке набор некоторых событий и аппаратных реакций на возникновение каждого из этих событий. Ситуация возникающая при наступлении такого события и сопровождающаяся временным или окончательным прекращением выполнения последовательности команд одной программы и переходом к выполнению команд другой программы, называется прерыванием. Аппарат прерываний используется, например, для управления внешними устройствами и для реализации возможности асинхронной работы с ними. Примером прерывания может служить прерывание по завершении обмена данными.

В момент возникновения прерывания действия аппаратуры вычислительной системы следующие:

- 1) В некоторые специальные регистры аппаратно заносится (сохраняется) информация о выполняемой в данный момент программе. Это минимальная информация, необходимая для начала обработки прерывания. Обычно в этот набор данных входит счетчик команд, регистр результата, указатель стека и несколько регистров общего назначения. Происходит так называемое малое "упрятывание".
- 2) В специальный управляющий регистр, иногда называемым регистром прерываний, помещается код возникшего прерывания.
- 3) Запускается программа обработки прерываний, входящая в состав операционной системы. Эта программа производит анализ причины прерывания. Если прерывание было фатальным (деление на ноль, например), то ОС прекращает выполнение программы, в которой возникло данное прерывание. Если же прерывание не фатальное, производится дополнительный анализ ситуа-

ции и делается вывод о том, можно ли оперативно обработать прерывание. Пример прерывания, которое всегда можно обработать оперативно - прерывание по таймеру. А прерывание, связанное, например, с приходом информации по линии связи, нельзя обработать оперативно, т.к. предварительно надо подкачать программу ОС, которая займется обработкой этого прерывания. При этом сначала происходит полное "упрятывание" - все регистры сохраняются в таблицах системы (а не в аппаратных регистрах, как раньше) и фиксируется то, что некоторые фрагменты программы, находящиеся в оперативной памяти, могут быть перенесены (при необходимости) на внешнее устройство, а затем производится обработка прерывания и возврат из прерывания.

Можно выделить следующие пять основных типов прерываний: внешние прерывания (при нажатии кнопки прерывания на пульте или при завершении интервала времени по сигналу таймера), прерывания при обращении к специальным функциям ОС, прерывания от схем контроля ЭВМ (при каких-либо сбоях аппаратуры), прерывания по вводу-выводу и программные прерывания (например, при делении на ноль, при передаче сигналов между программами).

Защита ресурсов и установка ограничений на использование ресурсов.

В вычислительной системе, работающей в режиме мультипрограммирования, возникает проблема защиты памяти, т.е. необходим реализованный на аппаратном уровне механизм, обеспечивающий защиту адресного пространства каждой из программ от несанкционированного доступа других программ.

Для вычислительных систем со страничной организацией памяти механизм защиты памяти может быть устроен, например, следующим образом: в таблице приписки, в которой отражено соответствие между физической и виртуальной памятью, в строках, относящихся к страницам, не принадлежащим данной программе или еще не загруженным в оперативную память, находится код меньше нуля. При попытке обратиться к такой странице в системе возникает прерывание по защите памяти. Обработывая это прерывание, ОС проверяет, действительно ли этой страницы памяти у данной программы нет, и, если эта страница чужая, то ОС прекращает выполнение данного процесса с диагностикой обращения в чужую память. Если же

какие то страницы еще не загружены в память, то ОС игнорирует прерывание, так как ошибки нет.

Современные многопользовательские ОС также решают задачу персонификации пользователей.

Персонификация - это возможность операционной системы идентифицировать конкретного пользователя и в соответствии с этим принимать те или иные действия, в частности, по защите данных.

Персонификация пользователей может быть организована, например, иерархически (аналогично иерархическим файловым системам). При этом существуют понятия "конкретный пользователь", "группа пользователей" и "все пользователи". Все пользователи делятся на группы, группы состоят из конкретных пользователей. Зная пользователя, группу, к которой он принадлежит, ОС имеет возможность определить и контролировать разные права доступа к ресурсам вычислительной системы для различных пользователей. Такая схема может быть многоуровневой (группы могут делиться на подгруппы и т.д.) с соответственным распределением прав и возможностей.

буферизация ввода/вывода.

По аналогии со способами борьбы с разницей в скорости доступа к различным компонентам вычислительной системы операционная система вводит в своих пределах программную буферизацию, которая также решает проблемы сглаживания времени доступа и проблемы синхронизации в целом. Сглаживание проблем, связанных с временем доступа, заключается в том, что практически каждая операционная система имеет КЭШ-буфера, которые аккумулируют обращения (образуют их очередь) к внешнему запоминающему устройству (ВЗУ) аналогично аппаратной буферизации при работе с оперативной памятью. Это позволяет существенно оптимизировать операционную систему. Признаком наличия такой буферизации является требование завершить выполнение операционной системы перед выключением машины. Степень этой буферизации определяет реальную эффективность системы.

20. Парадигмы программирования (функциональное, императивное, объектно-ориентированное программирование)

Таким образом, можно выделить две основные сущности процесса обработки информации: **код**, как совокупность инструкций, и **данные**. Все программы в соответствии с выбранной технологией программирования концептуально организованы вокруг своего кода или вокруг своих данных.

Рассмотрим основные на сегодняшний день парадигмы программирования:

1. **Процессно-ориентированная парадигма**, при которой программа представляет собой ряд последовательно выполняемых операций — модель фон-Неймана. При этом **код воздействует на данные**. Языки, реализующие эту парадигму, называются **процедурными** или **императивными**. Такими языками являются, например, C, Pascal и др.

Императивные языки. *Императивные, или процедурные, языки* — это управляемые командами или операторно-ориентированные языки программирования. Основной концепцией является состояние машины — множество всех значений всех ячеек памяти компьютера. Программа состоит из последовательности операторов, выполнение каждого из которых влечет за собой изменение значения в одной или нескольких ячейках памяти, то есть переход машины в новое состояние. В целом синтаксис такого языка имеет вид:

```
оператор1;  
оператор2;  
...
```

Рис. 1.1, а отображает этот процесс. Представим, что память состоит из набора *шариков в коробочках*, а выполнение оператора (например, сложение двух переменных для получения значения третьей) может быть представлено как обращение к ячейке памяти (коробочке), комбинирование тем или иным способом значений (шариков) и сохранение результата (нового шарика) в новой ячейке. Разработка программы состоит в построении последовательных состояний машины, необходимых для достижения результата. Обычно при первом знакомстве с концепциями программирования люди сталкиваются именно с этой моделью, и многие широко распространенные языки поддерживают именно ее (например, C, C++, FORTRAN, ALGOL, PL/1, Pascal, Ada, Smalltalk и COBOL). Эта модель вытекает из особенностей аппаратной части стандартного компьютера, выполняющей инструкции (команды) последовательно. Неудивительно, что большинство традиционных языков следуют именно этой модели.

1. **Аппликативная или функциональная парадигма.** Основная идея данного подхода заключается в формализованном определении функции, которую выполняет программа. Таким образом, вместо определения последовательности состояний, через которые должен пройти компьютер, чтобы получить требуемый результат, необходимо определить функцию, при применении которой к исходным данным получается требуемое решение:

$$\bar{y} = f(\bar{x})$$

Разработка программы при этом подходе сводится к конструированию сложной функции из имеющихся стандартных простых функций:

$$\bar{y} = f_1(f_2(f_3(\dots), f_4(\dots), \dots))$$

Языками, поддерживающими такую парадигму, являются, например, языки LISP и ML. Данные при таком подходе, так же, как и код, представ-

ляются списками одинаковой структуры, значит, программа, работая под управлением интерпретатора, может обрабатывать свой собственный код, как данные. в этом случае стирается грань между кодом и данными. Поэтому одной из важных областей применения данной парадигмы являются системы **искусственного интеллекта (ИИ)**.

Аппликативные языки. Другим взглядом на вычисления, производимые с помощью языка программирования, является рассмотрение функции, которую выполняет программа, а не отслеживание изменяемых состояний машины во время выполнения программы, оператор за оператором. Мы можем достичь этого, сосредоточив внимание на ожидаемом результате, а не на имеющихся в нашем распоряжении данных. Другими словами, вместо того чтобы рассматривать последовательность состояний, через которые должна пройти машина для получения ответа, вопрос следует поставить по-другому: какую функцию необходимо применить к начальному состоянию машины (путем выбора начального набора переменных и ком-

бинирования их определенным образом), чтобы получить требуемый ответ? Языки, в которых акцентирован именно этот взгляд на вычисления, называются *аппликативными*, или *функциональными*.

Эта модель представлена на рис. 1.1, б с помощью линзы, которая получает начальные данные и путем воздействия на их представление в памяти (как при прохождении света через линзу) производит желаемый ответ. Разработка программ сводится к созданию из уже имеющихся простых функций более сложных функций, которые поочередно воздействуют на начальный набор данных до тех пор пока последняя функция не выдаст требуемый результат. Вместо того чтобы рассматривать последовательные состояния машины, в данной модели мы концентрируем наше внимание на последовательных преобразованиях начальных данных с помощью функций. Как только у нас есть конечный ответ (последовательность функций), его можно применить к начальным данным и получить результат. Синтаксис такого языка, как правило, выглядит следующим образом:

функция_n(...функция₂(функция₁(данные))...)

Функциональное программирование предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы. Соответственно, не предполагает оно и изменимость этого состояния (в отличие от **императивного**, где одной из базовых концепций является **переменная**, хранящая своё значение и позволяющая менять его по мере выполнения алгоритма).

Концепции:

Функции высших порядков

Основная статья: [Функция высшего порядка](#)

Функции высших порядков — это такие функции, которые могут принимать в качестве аргументов и возвращать другие функции.^[12] Математики такую функцию чаще называют **оператором**, например, оператор взятия производной или оператор интегрирования.

Рекурсия

Основная статья: [Рекурсия](#)

В функциональных языках **цикл** обычно реализуется в виде рекурсии. Строго говоря, в функциональной парадигме программирования нет такого понятия, как цикл.

Рекурсивные функции вызывают сами себя, позволяя операции выполняться снова и снова. Для использования рекурсии может потребоваться большой **стек**, но этого можно избежать в случае **хвостовой рекурсии**. Хвостовая рекурсия может быть распознана и оптимизирована компилятором в код, получаемый после компиляции аналогичной итерации в императивном языке программирования.

2. Парадигма, основанная на использовании **системы правил (парадигма логического программирования)**. При этом подходе операторы программы выполняются не в той последовательности, в которой они написаны, а на основе анализа **разрешающих условий (РУ)**.

Программа при такой парадигме состоит из списка пар:

$$\begin{aligned} PУ_1 &\rightarrow D_1 \\ PУ_2 &\rightarrow D_2 \\ &\dots\dots\dots \\ PУ_N &\rightarrow D_N \end{aligned}$$

Здесь $D_1, D_2, \dots, D_N$ — действия, выполняемые в случае истинности соответствующих разрешающих условий $PУ_1, PУ_2, \dots, PУ_N$.

Выполнение программы заключается в циклической проверке разрешающих условий и выполнения действий, соответствующих разрешающим условиям, в случае истинности последних.

Примером языка логического программирования является язык PROLOG.

2. **Объектно-ориентированная парадигма**, при которой программа рассматривается как совокупность фрагментов кода, обрабатывающих отдельные совокупности данных — **объекты**. Эти объекты взаимодействуют друг с другом посредством так называемых **интерфейсов**. При этом данные управляют доступом к коду.

Центральной идеей ООП является реализация понятия «**абстракция**». Смысл абстракции заключается в том, что *сущность* произвольной сложности можно рассматривать, а также производить определенные действия над ней, как над **единым целым**, не вдаваясь в детали внутреннего построения и функционирования.

При создании программного комплекса необходимо разработать определенные абстракции.

Одним из основных способов создания абстракции является использование концепции **иерархической классификации**. Ее суть заключается в том, что сложные системы разбиваются на более простые фрагменты.

Практически все сложные системы иерархичны, и уровни их иерархии отражают различные уровни абстракции. Для каждой конкретной задачи рассматривается соответствующий уровень. Выбор низшего уровня абстракции достаточно произволен. Выбранный уровень в одном случае в качестве низшего уровня может оказаться уровнем достаточно высокой абстракции в другом проекте.

Различают **типовую** иерархию и **структурную** иерархию, которые далее мы будем называть соответственно **структурой классов** и **структурой объектов**.

Во всех объектно-ориентированных языках программирования реализованы следующие **основные механизмы (постулаты) ООП**:

- **инкапсуляция,**
- **наследование,**
- **полиморфизм.**

Все эти механизмы важны для разработки и использования абстракций.

1) Инкапсуляция — механизм, связывающий вместе код и данные, которыми он манипулирует, и одновременно защищающий их от произвольного доступа со стороны другого кода, внешнего по отношению к рассматриваемому. Доступ к коду и данным жестко контролируется интерфейсом.

Основой инкапсуляции при ООП является **класс**.

Механизм инкапсуляции позволяет оставлять скрытыми от пользователя некоторые детали реализации класса (то есть инкапсулировать их в классе), что упрощает работу с объектами этого класса.

2) Наследование — механизм, с помощью которого один объект (производного класса) приобретает свойства другого объекта (родительского, базового класса). При использовании наследования новый объект не обязательно описывать, начиная с нуля, что существенно упрощает работу программиста. Наследование позволяет какому-либо объекту наследовать от своего родителя общие атрибуты, а для себя определять только те характеристики, которые делают его уникальным внутри класса.

Наследование есть очень важное понятие, поддерживающее концепцию иерархической классификации.

3) Полиморфизм — механизм, позволяющий использовать один и тот же интерфейс для общего класса действий.

Вместо трех подпрограмм управления в объектно-ориентированной программе требуется всего одна подпрограмма (один интерфейс)

Общая концепция полиморфизма: **один интерфейс — много методов.**

Выбор конкретного действия (метода) применительно к конкретной ситуации возлагается на компилятор. Программисту же достаточно запомнить и применить один интерфейс, вместо нескольких, что также упрощает работу.

Различаются **статический** (реализуется на этапе компиляции с помощью перегрузки функций и операций), **динамический** (реализуется во время выполнения программы с помощью механизма виртуальных функций) и **параметрический** (реализуется на этапе компиляции с использованием механизма шаблонов) полиморфизм.

Типы данных, создаваемые пользователем (программистом), называются **пользовательскими типами данных**. Пользовательский тип данных с полностью скрытой (инкапсулированной) внутренней структурой называется **абстрактным типом данных (АТД)**.

21. Базы данных. Основные понятия реляционной модели данных. Реляционная алгебра. Средства языка запросов SQL.

В широком спектре автоматизированных информационных систем особо выделяются системы, предназначенные для поддержки удобного и долговременного хранения и обработки больших объемов информации, имеющей, как правило, достаточно сложную структуру. Для хранения такого рода информации на практике используются базы данных (БД), при этом под *базой данных* в общем случае понимается набор любых систематизированных данных, согласованных между собой. Обращение к базе данных и управление ею осуществляется с помощью *системы управления базами данных* (СУБД), которая представляет собой программную систему, ориентированную на поддержку хранения, манипулирования и управления данными, обеспечивая при этом их целостность и безопасность.

Способ организации информации в базе данных определяется конкретной *моделью данных*, положенной в основу построения этой базы. Модель данных описывает на концептуальном уровне связи между элементами хранимых данных и допустимые операции над ними, т.е. задает логическую структуру данных и механизм их обработки. Так иерархическая модель базируется на организации данных в виде деревьев, сетевая использует в качестве основного способа представления данных графы и сети.

4. Основные функции СУБД, типовая организация СУБД.

1) Непосредственное управление данными во внешней памяти

Эта функция обеспечивает *поддержку необходимых структур внешней памяти* как для хранения данных и метаданных, непосредственно входящих в БД, так и для служебных целей, например, для ускорения доступа к данным (индексы).

2) Управление буферами оперативной памяти.

СУБД обычно работают с БД значительного размера; по крайней мере, этот размер обычно существенно больше объема доступной ОП.

3) Управление транзакциями.

Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое:

4) Журнализация.

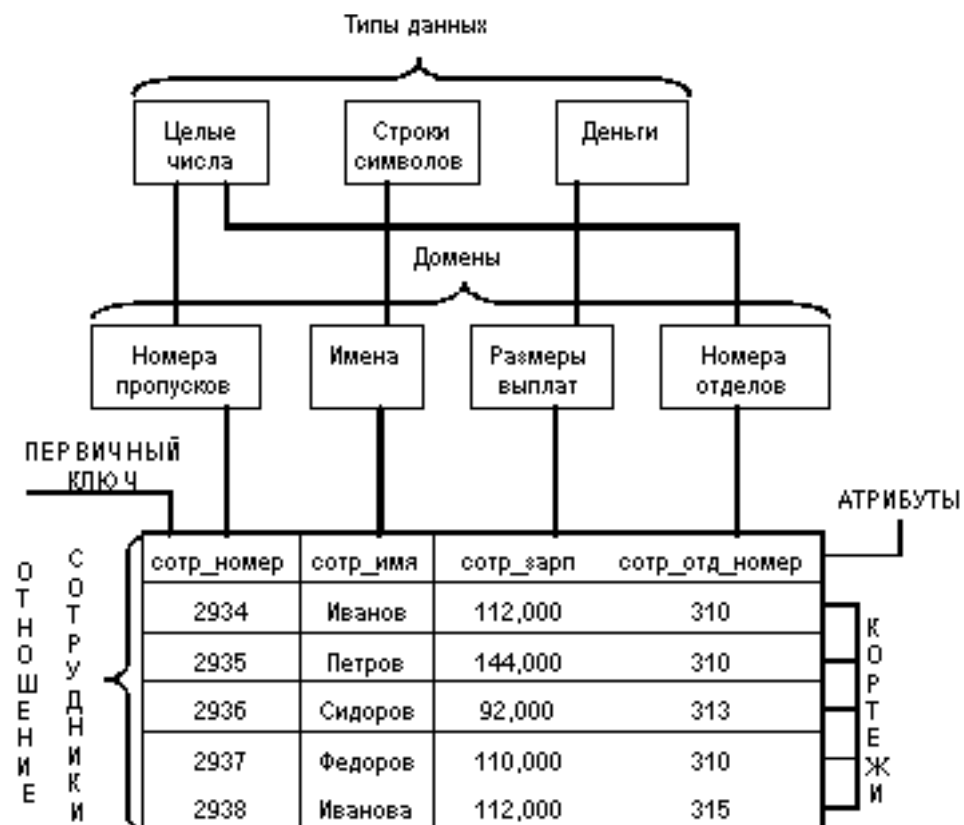
Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя.

5) Поддержка языков БД

Для работы с базами данных используются специальные языки, в целом называемые **языками баз данных**. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка: **язык определения схемы БД** (*SDL - Schema Definition Language*) и **язык манипулирования данными** (*DML - Data Manipulation Language*). SDL служил, главным образом, для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные.

Основными понятиями реляционных баз данных являются **тип данных**, **домен**, **атрибут**, **кортеж**, **первичный ключ** и **отношение**.

Для начала покажем смысл этих понятий на примере отношения СОТРУДНИКИ, содержащего информацию о сотрудниках некоторой организации:



Понятие **тип данных** в реляционной модели данных полностью адекватно понятию типа данных в языках программирования. Определение типа данных состоит из **трех основных компонентов**: множества значений данного типа; набор операций, применимых к значениям типа; способ внешнего представления значений типа (литералов)

Обычно в современных реляционных БД допускается хранение символьных, числовых данных (точных и приближительных), битовых строк, специализированных числовых данных (таких как "деньги"), а также специальных "темпоральных" данных (дата, время, временной интервал). Достаточно активно развивается подход к расширению возможностей реляционных систем абстрактными типами данных (соответствующими возможностями обладают, например, системы семейства Ingres/Postgres). В нашем примере мы имеем дело с данными трех типов: строки символов, целые числа и "деньги".

Понятие **домена** более специфично для баз данных, хотя и **имеет некоторые аналогии с подтипами в некоторых языках программирования**. В самом общем виде домен **определяется** заданием некоторого **базового типа данных**, к которому относятся элементы домена, и произвольного **логического выражения**, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат "**истина**", то элемент данных является элементом домена. С каждым доменом связывается имя, уникальное среди имен всех доменов соответствующей базы данных.

Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального ограниченного подмножества значений данного типа. Например, домен "Имена" в нашем примере определен на базовом типе строк символов, но в число его значений могут входить только те строки, которые могут изображать имя (в частности, такие строки не могут начинаться с мягкого знака).

Если некоторый атрибут отношения определяется на некотором домене (как, например, атрибут сотр_имя определяется на домене имени), то в дальнейшем **ограничение домена играет роль ограничения целостности**, накладываемого на значения этого атрибута

Следует отметить также семантическую нагрузку понятия домена: **данные считаются сравнимыми только в том случае, когда они относятся к одному домену**. В нашем примере значения доменов "Номера пропусков" и "Номера групп" относятся к типу целых чисел, но не являются сравнимыми.

Отношение.

Для уточнения термина *отношение* выделяются понятия **заголовка отношения**, **значения отношения** и **переменной отношения**. Кроме того, требуется вспомогательное понятие **кортежа**.

Заголовком (или схемой) отношения R (HR) называется конечное множество упорядоченных пар вида $\langle A, T \rangle$, где A называется именем *атрибута*, а T – имя некоторого базового типа или ранее определенного домена. По определению требуется, чтобы **все имена атрибутов в заголовке отношения были различны**. В примере

заголовком отношения СЛУЖАЩИЕ является множество пар {<сотр_номер номера пропусков>, <сотр_имя, имена>, <сотр_зарп, Размеры выплат>, <сотр_отд_номер, номера отделов>}.

Если все атрибуты заголовка отношения определены на разных доменах, то, чтобы не плодить лишних имен, разумно использовать для именования атрибутов имена соответствующих доменов.

Кортежем tR , соответствующим заголовку HR , называется множество упорядоченных триплетов вида $\langle A, T, v \rangle$, по одному такому триплету для каждого атрибута в HR . Третий элемент – v – триплета $\langle A, T, v \rangle$ должен являться допустимым значением типа данных или домена T

Заголовку отношения СЛУЖАЩИЕ соответствуют, например, следующие кортежи: {<сотр_номер, номера пропусков, 2934>, <сотр_имя, имена, Иванов>, <сотр_зарп, размеры выплат, 112.000>, <сотр_отд_номер, Номера отделов, 310>}

Телом BR отношения R называется произвольное множество кортежей tR . Одно из возможных тел отношения Сотрудники показано на рисунке. Заметим, что в общем случае могут существовать такие кортежи tR , которые соответствуют HR , но не входят в BR . **Значением** VR отношения R называется пара множеств HR и BR . Одно из допустимых значений отношения СОТРУДНИКИ показано на рисунке.

В изменчивой реляционной базе данных хранятся отношения, значения которых изменяются во времени. **Переменной** $VARR$ называется именованный контейнер, который может содержать любое допустимое значение VR . Естественно, что при определении любой $VARR$ требуется указывать соответствующий заголовок отношения HR .

Заметим, что в дальнейшем в тех случаях, когда точный смысл термина понятен по контексту, мы будем использовать не уточненный термин **отношение**, как в смысле **значение отношения**, так и в смысле **переменная отношения**.

По определению **степенью**, или **“арностью”** заголовка отношения, кортежа, соответствующего этому заголовку, тела отношения, значения отношения и переменной отношения является мощность заголовка отношения

Например, степень отношения СОТРУДНИКИ равна четырём, то есть оно является 4-арным (*кватернарным*). Разумно считать **схемой реляционной базы данных** набор пар $\langle \text{имя_}VARR, HR \rangle$, включающий имена и заголовки

всех переменных отношения, которые определены в базе данных

Реляционная база данных – это набор пар $\langle VARR, Hr \rangle$ (конечно, каждая переменная отношения в любой момент времени содержит некоторое значение-отношение, в частности, пустое).

Заметим, что в классических реляционных базах данных после определения схемы базы данных могли изменяться только значения переменных отношений. Однако теперь в большинстве реализаций допускается и изменение схемы базы данных: определение новых и изменение заголовков существующих переменных отношений. Это принято называть **эволюцией** схемы базы данных.

По определению, **первичным ключом** переменной отношения является такое подмножество S множества атрибутов ее заголовка, что в любое время значение первичного ключа (составное, если в состав первичного ключа входит более одного атрибута) в любом кортеже тела отношения отличается от значения первичного ключа в любом другом кортеже тела этого отношения, а никакое собственное подмножество S этим свойством не обладает. Существование первичного ключа у любого значения отношения является следствием одного из фундаментальных свойств отношений, а именно того свойства, что тело отношения является множеством кортежей.

Обычным житейским представлением отношения является *таблица*, заголовком которой является схема отношения, а *строками* – кортежи отношения-экземпляра; в этом случае имена атрибутов соответствуют именам *столбцов* данной таблицы. Поэтому иногда говорят про «столбцы таблицы», имея в виду «атрибуты отношения».

То свойство, что тело любого отношения никогда не содержит кортежей-дубликатов, следует из определения тела отношения как множества кортежей. В классической теории множеств по определению любое множество состоит из различных элементов.

Именно из этого свойства вытекает наличие у каждого значения отношения **первичного ключа** – минимального множества атрибутов, являющегося подмножеством заголовка данного отношения, составное значение которых уникально определяет кортеж отношения. Действительно, поскольку в любое время все кортежи тела любого отношения различны, у любого значения отношения свойством уникальности обладает, по крайней мере, полный набор его атрибутов. Однако в формальном определении первичного ключа требуется обеспечение его «минимальности», т. е. в набор атрибутов первичного ключа не должны входить такие атрибуты, которые можно отбросить без ущерба для основного свойства – однозначного определения кортежа. Немного позже мы покажем, почему свойство минимальности первичного ключа является критически важным. Понятно, что **если у любого отношения существует набор атрибутов, обладающий свойством уникальности, то существует и минимальный набор атрибутов, обладающий свойством уникальности.**

атрибутов назвать первичным ключом, а остальные минимальные наборы атрибутов, обладающие свойством уникальности, называются *возможными ключами*.

Значения всех атрибутов являются атомарными (вернее, скалярными). Это следует из **определения домена как потенциального множества значений скалярного типа данных**, т. е. среди значений домена не могут содержаться значения с видимой структурой, в том числе множества значений (отношения). Заметим, что это не противоречит потенциальной возможности использования при спецификации атрибутов типов данных, определяемых пользователями. Например, можно было бы добавить в схему отношения СЛУЖАЩИЕ атрибут СЛУ_ФОТО, определенный на домене (или типе данных) ФОТОГРАФИИ. Главное в атомарности значений атрибутов состоит в том, что реляционная СУБД не должна обеспечивать пользователям явной **видимости внутренней структуры значения**. Со всеми значениями можно обращаться

только с помощью операций, определенных в соответствующем типе данных.

В манипуляционной части модели определяются два фундаментальных механизма манипулирования реляционными БД – **реляционная алгебра и реляционное исчисление**. Первый механизм базируется в основном на **классической теории множеств** (с некоторыми уточнениями и добавлениями), а второй – на **классическом логическом аппарате исчисления предикатов первого порядка**. Основной функцией манипуляционной части реляционной модели является **обеспечение меры реляционности любого конкретного языка реляционных БД**: язык называется реляционным, если он обладает не меньшей выразительностью и мощностью, чем реляционная алгебра или реляционное исчисление.

В целостной части реляционной модели данных фиксируются два базовых требования целостности, которые должны поддерживаться в любой реляционной СУБД. Первое требование называется **требованием целостности сущностей**. Объекту или сущности реального мира в реляционных БД соответствуют кортежи отношений. Конкретно требование состоит в том, что **любой кортеж любого отношения отличим от любого другого кортежа этого отношения**, т.е. другими словами, **любое отношение должно обладать первичным ключом**. Это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

На самом деле, требование целостности сущности полностью звучит следующим образом: **у любой переменной отношения должен существовать первичный ключ, и никакое значение первичного ключа в кортежах значения-отношения переменной отношения не должно содержать неопределенных значений**.

Неопределенное значение не принадлежит никакому типу данных и может присутствовать среди значений любого атрибута, определенного на любом типе данных (если это явно не запрещено при определении атрибута). Если a – это значение некоторого типа данных или NULL, op – любая двуместная «арифметическая» операция этого типа данных (например, +), а lop – операция сравнения значений этого типа (например, =), то по определению:

$a \text{ } op \text{ } NULL = NULL$
 $NULL \text{ } op \text{ } a = NULL$
 $a \text{ } lop \text{ } NULL = unknown$
 $NULL \text{ } lop \text{ } a = unknown$

Здесь *unknown* – это третье значение логического, или булевского, типа, обладающее следующими свойствами:

$NOT \text{ } unknown = unknown$
 $true \text{ } AND \text{ } unknown = unknown$
 $true \text{ } OR \text{ } unknown = true$
 $false \text{ } AND \text{ } unknown = false$
 $false \text{ } OR \text{ } unknown = unknown$

Второе требование называется **требованием целостности по ссылкам** и является несколько более сложным. Очевидно, что при соблюдении нормализованности отношений сложные сущности реального мира представляются в реляционной БД в виде нескольких кортежей нескольких отношений.

Атрибут называется **внешним ключом**, если его значения однозначно характеризуют сущности, представленные кортежами некоторого другого отношения (т.е. задают значения их первичного ключа). Говорят, что **отношение, в котором определен внешний ключ, ссылается на соответствующее отношение, в котором такой же атрибут является первичным ключом**.

Требование целостности по ссылкам, или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа, появляющегося в ссылающемся отношении, в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть неопределенным (т.е. ни на что не указывать).

Здесь существуют **три подхода**, каждый из которых поддерживает целостность по ссылкам. Первый подход заключается в том, что **запрещается производить удаление кортежа, на который существуют ссылки** (т.е. сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа). При втором подходе при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах **значение внешнего ключа автоматически становится неопределенным**. Наконец, третий подход (каскадное удаление) состоит в том, что **при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи**.

Основная идея реляционной алгебры состоит в том, что коль скоро отношения являются множествами, **средства манипулирования отношениями могут базироваться на традиционных теоретико-множественных операциях, дополненных некоторыми специальными операциями**, специфичными для реляционных баз данных.

Существует много подходов к определению реляционной алгебры, которые различаются наборами операций и способами их интерпретации, но, в принципе, являются более или менее равносильными. В этом варианте набор основных алгебраических операций состоит из восьми операций, которые делятся на **два класса** – теоретико-множественные операции и специальные реляционные операции. В состав **теоретико-множественных операций** входят операции:

- объединения отношений;
- пересечения отношений;
- взятия разности отношений;
- взятия декартова произведения отношений

Специальные реляционные операции включают:

- ограничение отношения;
- проекцию отношения;
- соединение отношений;
- деление отношений.

Кроме того, в состав алгебры включается **операция присваивания**, позволяющая сохранить в базе данных результаты вычисления алгебраических выражений, и **операция переименования атрибутов**, дающая возможность корректно сформировать заголовок (схему) результирующего отношения.

Общая интерпретация реляционных операций

- При выполнении операции **объединения** (UNION) двух отношений с **одинаковыми заголовками** производится отношение, включающее все кортежи, которые входят хотя бы в одно из отношений-операндов.
- Операция **пересечения** (INTERSECT) двух отношений с **одинаковыми заголовками** производит отношение, включающее все кортежи, которые входят в оба отношения-операнда.
- Отношение, являющееся **разностью** (MINUS) двух отношений с **одинаковыми заголовками**, включает все кортежи, входящие в отношение-первый операнд, такие, что ни один из них не входит в отношение, которое является вторым операндом.
- При выполнении **декартова произведения** (TIMES) двух отношений, **пересечение заголовков которых пусто**, производится отношение, кортежи которого производятся путем объединения кортежей первого и второго операндов.
- Результатом **ограничения** (WHERE) отношения по некоторому условию является отношение, включающее кортежи отношения-операнда, удовлетворяющее этому условию.
- При выполнении **проекции** (PROJECT) отношения на заданное подмножество множества его атрибутов производится отношение, кортежи которого являются соответствующими подмножествами кортежей отношения-операнда.
- При **соединении** (JOIN) двух отношений по некоторому условию образуется результирующее отношение, кортежи которого производятся путем объединения кортежей первого и второго отношений и удовлетворяют этому условию.

$(A \text{ JOIN } B \text{ WHERE } \text{comp} = (A \text{ TIMES } B) \text{ WHERE } \text{comp})$

Имеются важный частный случай соединения – **эквисоединение** (EQUIJOIN) и простое, но важное расширение операции эквисоединения – **естественное соединение** (NATURAL JOIN) Операция соединения называется операцией эквисоединения, если условие соединения имеет вид $(a = b)$, где a и b – атрибуты разных операндов соединения.

Операция естественного соединения применяется к паре отношений A и B , обладающих (возможно составным) общим атрибутом c (т.е. атрибутом с одним и тем же именем и определенным на одном и том же домене) Пусть ab обозначает объединение заголовков отношений A и B . Тогда естественное соединение A и B – это спроецированный на ab результат эквисоединения A и B по условию $A.c = B.c$ Хотя операция естественного соединения выражается через операции переименования, соединения общего вида и проекции, для нее обычно используется сокращенная форма, называемая NATURAL JOIN

- **Реляционное деления (DIVIDE BY)**. Пусть заданы два отношения – A с заголовком $\{a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m\}$ и B с заголовком $\{b_1, b_2, \dots, b_m\}$. Будем считать, что атрибут b_i отношения A и атрибут b_i ($i = 1, 2, \dots, m$) отношения B не только обладают одним и тем же именем, но и определены на одном и том же домене

Назовем множество атрибутов $\{a_j\}$ составным атрибутом a , а множество

атрибутов $\{b_j\}$ – составным атрибутом b . После этого будем говорить о

реляционном делении бинарного отношения $A \{a, b\}$ на унарное отношение $B \{b\}$

Результатом деления A на B ($A \text{ DIVIDE BY } B$) является унарное отношение C

$\{a\}$, тело которого состоит из кортежей v таких, что в теле отношения A

содержатся кортежи $\langle v, w \rangle$ такие, что множество значений $\{w\}$ включает

множество значений атрибута b в отношении B

- Операция **переименования (RENAME)** производит отношение, тело которого совпадает с телом операнда, но имена атрибутов изменены.
- Операция **присваивания ($:=$)** позволяет сохранить результат вычисления реляционного выражения в существующем отношении БД.

Операция	Приоритет
RENAME	4
WHERE	3
PROJECT	3
TIMES	2
JOIN	2
INTERSECT	2
DIVIDE BY	2
UNION	1
MINUS	1

Эти соображения подводят к понятию **совместимости отношений по объединению**:

два отношения совместимы по объединению в том и только в том случае, когда

обладают одинаковыми заголовками. В развернутой форме это означает, что в

заголовках обоих отношений содержится один и тот же набор имен атрибутов, и

одноименные атрибуты определены на одном и том же домене (эта развернутая

формулировка, вообще говоря, является излишней, но она пригодится нам в следующем абзаце).

Поэтому в реляционной алгебре используется специализированная форма операции взятия декартова произведения – **расширенное декартово произведение отношений**. При взятии расширенного декартова произведения двух отношений элементом результирующего отношения является кортеж, который представляет собой объединение одного кортежа первого отношения и одного кортежа второго отношения.

Эти соображения приводят к введению понятия **совместимости по взятию расширенного декартова произведения**. Два отношения **совместимы по взятию расширенного декартова произведения** в том и только в том случае, если пересечение множеств имен атрибутов, взятых из их схем отношений, пусто. Любые два отношения всегда могут стать совместимыми по взятию декартова произведения, если применить операцию переименования к одному из этих отношений.

По поводу теоретико-множественных операций реляционной алгебры следует еще заметить, что **все четыре** операции являются **ассоциативными**. **Все операции, кроме взятия разности**, являются **коммутативными**, т. е. $A \text{ OP } B = B \text{ OP } A$.

SQL — универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных.

SQL основывается на реляционной алгебре.

Язык SQL представляет собой совокупность

- операторов;
- инструкций;
- и вычисляемых функций.

Операторы

Согласно общепринятому стилю программирования, операторы (и другие зарезервированные слова) в SQL всегда следует писать прописными буквами.

Операторы SQL делятся на:

- операторы определения данных (*Data Definition Language, DDL*)
 - **CREATE** создает объект БД (саму базу, таблицу, представление, пользователя и т. д.)
 - **ALTER** изменяет объект
 - **DROP** удаляет объект
- операторы манипуляции данными (*Data Manipulation Language, DML*)
 - **SELECT** считывает данные, удовлетворяющие заданным условиям
 - **INSERT** добавляет новые данные
 - **UPDATE** изменяет существующие данные
 - **DELETE** удаляет данные
- операторы определения доступа к данным (*Data Control Language, DCL*)
 - **GRANT** предоставляет пользователю (группе) разрешения на определенные операции с объектом
 - **REVOKE** отзывает ранее выданные разрешения
 - **DENY** задает запрет, имеющий приоритет над разрешением
- операторы управления транзакциями (*Transaction Control Language, TCL*)
 - **COMMIT** применяет транзакцию.
 - **ROLLBACK** откатывает все изменения, сделанные в контексте текущей транзакции.
 - **SAVEPOINT** делит транзакцию на более мелкие участки.

Создание базы данных.

```
CREATE DATABASE zawod
```

```
CREATE TABLE ceh ( nomerceh int, namesceh char(20) )
```

В уже существующей таблице мы можем поменять тип столбца, добавить новый, уничтожить старый.

```
-----  
ALTER TABLE kadry ADD (dolvnostx CHAR(20) BEFORE zarplata),  
DROP(pribytie), ADD CONSTRAINT UNIQUE(tabnom, fio) CONSTRAINT  
tabnomfio
```

```
ALTER TABLE items MODIFY (manu_code char(4))
```

Изменение структуры таблицы приводит к физическому преобразованию данных в ней. Если изменен тип столбца, то данные в нем преобразуются к новому типу, и если это невозможно осуществить, то оператор ALTER "валится" с кодом ошибки, а таблица остается в неизменном состоянии.

View - "псевдо" таблица, базируется на существующих таблицах.

```
-----  
CREATE VIEW poor AS SELECT tabnom, fio, datarovd FROM kadry  
WHERE zarplata < 120  
# создано view - "псевдотаблица" из трех столбцов содержащая  
# строки из таблицы kadry, в которых zarplata меньше 120 рублей.
```

Ведет себя точно так же, как настоящая таблица, только место на диске под нее не отводятся, поскольку данные, лежащие в ней на самом деле хранятся в таблице, на которой это view базируется.

Индекс - дополнительная структура к столбцам таблицы, нужен для ускорения поиска значений в столбце.

Выдавать и забирать права доступа к таблице может владелец таблицы, Администратор Базы Данных (имеющий DBA права), а так же пользователь, которому было выдано право выдавать права (Оператором GRANT WITH GRANT OPTIONS)

```
-----  
REVOKE ALL ON customer FROM PUBLIC  
GRANT ALL ON customer TO iwanow, petrow WITH GRANT OPTION
```

```
GRANT UPDATE(fname,lname,company, sity),SELECT  
ON customer TO PUBLIC
```

```
REVOKE CONNECT FROM sidorowa, root  
REVOKE DBA FROM ivanov
```

Отобрать у вас права DBA (если вы, конечно, им являетесь) может только другой DBA.

На время транзакции все измененные строки автоматически блокируются системой от изменения (но не от просмотра). Вы можете явно локировать всю таблицу целиком, тогда система не будет блокировать строки по отдельности. Вы можете блокировать таблицу целиком не только от изменения но и от просмотра.

```
-----  
BEGIN WORK
```

```

LOCK TABLE kadry
. . .
UNLOCK TABLE kadry
. . .
LOCK TABLE kadry EXCLUSIVE
-----

```

Если ваш оператор пытается записать в блокированную другим пользователем строку, то оператор "сваливается". Вы можете установить для своей программы режим "Ждать разблокирования строк".

```

-----
SET LOCK MODE TO WAIT
-----

```

Указать базе новый системный журнал.

```

-----
START DATABASE завод WITH LOG IN "/ARM/log/завод"
-----

```

Восстанавливают разрушенную (например из-за сбоя оборудования) базу данных так:

В среде UNIX

1. Уничтожают остатки базы.
 2. Заливают вместо них копию базы с ленты (например, недельной давности) используя команду `cpio` или утилиту `INFORMIX dbimport`.
 3. Входят в среду `INFORMIX` и выполняют операторы:

<code>DATABASE kadry EXCLUSIVE</code>	<code># чтобы никто не лез</code>
<code>ROLLFORWARD DATABASE kadry</code>	<code># прогнать базу вперед</code>
	<code># по системному журналу</code>
<code>CLOSE DATABASE</code>	<code># теперь всем можно работать</code>
-

Транзакция

```

-----
BEGIN WORK      # начать транзакцию
. . .          # операторы

IF все нормально THEN COMMIT WORK
ELSE             ROLLBACK WORK
END IF
-----

```

Если во время транзакции программа "свалилась" то `INFORMIX` автоматически сделает откатку.

• Простейшая форма оператора **SELECT**.

-
- Первый пример находит в таблице `kadry` строку, в которой столбец `tabnum=345`. Из этой строки берутся только три указанных столбца.
- Второй пример выбирает ВСЕ строки из таблицы `ceh`, и все столбцы.

```

-----
• SELECT fio, dolvn, zarplata FROM kadry WHERE tabnom=345
•
• SELECT * FROM ceh
•

```

- `SELECT kadry.fio, ceh.nameceh WHERE kadry.nomerceh=ceh.nomerceh`
- -----
- Третий пример выбирает фамилии работников из таблицы кадры, а названия цехов, в которых они работают, из таблицы ceh.
- -----

```
INSERT INTO kadry VALUES (4,0,"Грицько",num,"10/25/1939",NULL)
```

```
INSERT INTO customer VALUES (ps_customer.*)
```

ps_customer – переменная типа RECORD – аналог структуры в языке Си. Этот оператор вставляет значения элементов записи

ps_customer в соответствующие поля таблицы customer

```
INSERT INTO kadry      (tabnom, fio, nomerceh, dolvnostx)
      SELECT  0 , fio, 4, dolvnostx FROM kadryold
      WHERE nomerceh=3 AND fio IS NOT NULL
```

последний оператор вставляет сразу несколько строк

- **UPDATE.**
-
- меняет значения столбцов, в строках, удовлетворяющим WHERE ус-
- ловию.
- -----
- `UPDATE kadry SET fio="Зыкова" WHERE fio="Гирусова"`
-
- `UPDATE ceh SET kod_ceha[1,4]=nameceh[5,8] WHERE`
- `nomerceh BETWEEN 3 AND 5 OR nameceh IN ("токарный", "литейный")`
- -----
- В таблице ceh в цехах номер 3,4,5 а так же в токарном и литей-
- ном первые четыре символа в коде цеха будут заменены на подст-
- року поля nameceh из той же строки.

Предложение WHERE может присутствовать в любом из операторов DELETE, UPDATE, SELECT, когда нужно задать условия на строки, которые требуется обработать (соответственно, уничтожить, изменить или выбрать). Рассмотрим структуру и примеры использования предложений WHERE.

- *** УСЛОВИЯ С ПОДЗАПРОСОМ ***
-
- -----
- `SELECT fio FROM kadry WHERE zarplata=`
- `(SELECT MAX(zarplata) FROM kadry )`
- -----
- Здесь подзапрос возвращает единственное значение – максимальное
- значение зарплаты. А внешний SELECT оператор находит фамилии
- обладателей оной.
-
- -----
- `SELECT fio, shifr, organizaciq FROM zaqwki WHERE denxgi_rek is`
- `not NULL and`
- `gorod in (SELECT gorod FROM regiony WHERE region="Урал")`
- -----
- Здесь запрос выводит данные об руководителях, получивших финан-
- сирование и работающих на Урале.
-
- -----

- `SELECT order_num, stock_num, manu_code, total_price FROM items x`
- `WHERE total_price > (SELECT 2*MIN(total_price)`
- `FROM items WHERE order_num=x.order_num)`

- Этот запрос (используя связанный подзапрос) выводит список всех изделий, чья общая цена не менее чем в два раза превосходит минимальную цену изделий перечисленных в этом же ордере.

• Вы можете соединять любое количество вышеперечисленных условий вместе, используя логические операторы NOT, AND, OR.

• Оператор **UNLOAD**

- Оператор UNLOAD сбрасывает данные из таблицы в файл в печатном представлении. Каждая строка преобразуется в отдельную запись, значения из столбцов разделяются символом " | ".

- После выполнения оператора

- `UNLOAD TO "kadry19.unl" SELECT * FROM kadry`

- в файле kadry19.unl можно будет обнаружить следующее:

- 5 | 5 | туев | завхоз | 100.0 | 31.12.1946 |
- 4 | 6 | петунин | кладовщик | 80.0 | |

- . . .

• Оператор **LOAD**

- Оператор LOAD выполняет обратную операцию – считывает строки из файла и вставляет их в таблицу. Естественно, что типы и количество значений в строках файла должны соответствовать столбцам таблицы.

- `LOAD FROM "kadry20.unl" INSERT INTO kadry`

Группировка используется для для "сплющивания" группы (строк) в одну.

Результат запроса содержит одну строку для каждого множества строк, удовлетворяющих WHERE предложению и содержащих одно и то же значение в указанном столбце.

- `SELECT dolvnostx, COUNT(*), AVG(zarplata) FROM kadry`
`GROUP BY dolvnostx`

Строки из таблицы, присоединенной внешним образом (на внешнее соединение указывает ключевое слово OUTER) будут выбираться не смотря на то, удовлетворяют они условиям WHERE предложения или нет. В некоторых случаях это полезно, когда у вас есть главная таблица и есть вспомогательная, и данные из главной таблицы вам нужно получить в любом случае. Пример внешнего соединения:

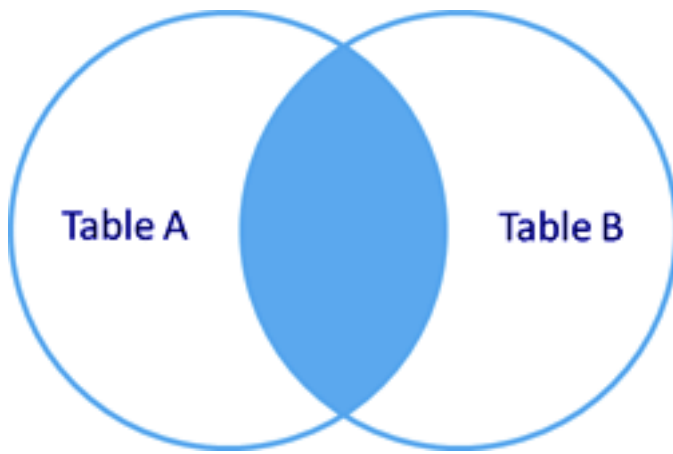


рис. Inner join

Внутреннее объединение INNER JOIN (синоним JOIN, ключевое слово INNER можно опустить).

Выбираются только совпадающие данные из объединяемых таблиц. Чтобы получить данные, которые не подходят по условию, необходимо использовать

внешнее объединение - OUTER JOIN.

Такое объединение вернет данные из обеих таблиц совпадающими по одному из условий.

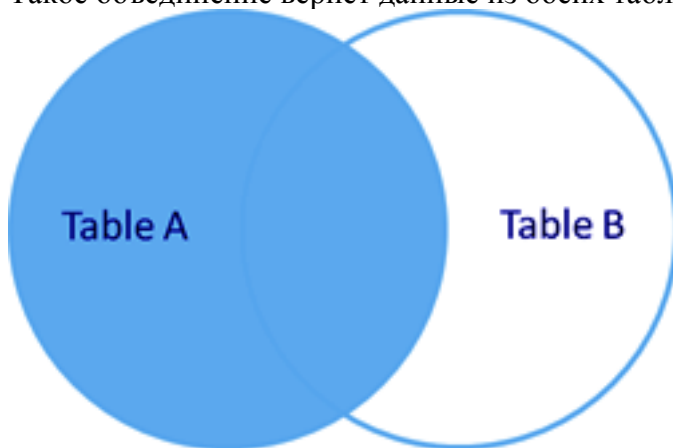


рис. Left join

Существует два типа внешнего объединения OUTER JOIN - LEFT OUTER JOIN и RIGHT OUTER JOIN.

Работают они одинаково, разница заключается в том что LEFT - указывает что "внешней" таблицей будет находящаяся слева (в нашем примере это таблица users).

Ключевое слово OUTER можно опустить. Запись LEFT JOIN идентична LEFT OUTER JOIN.

```
SELECT u.id, u.name, d.name AS d_name  
FROM users u  
LEFT OUTER JOIN departments d ON u.d_id = d.id
```

Транзакция начинается всякий раз, когда вы начинаете сеанс с SQL. Все команды которые вы введете будут частью этой транзакции, пока вы не завершите их вводом команды **COMMIT WORK** или команды **ROLLBACK WORK**. **COMMIT** может сделать все изменения постоянными с помощью транзакции, а **ROLLBACK** может откатить их обратно или отменить. Новая транзакция начинается после каждой команды COMMIT или ROLLBACK. Этот процесс известен как *диалоговая обработка запросов* или *транзакция*. Синтаксис, чтобы оставить все ваши изменения постоянными во время регистрации, или во время последнего COMMIT или ROLLBACK

```
COMMIT WORK;
```

Синтаксис отмены изменения -

```
ROLLBACK WORK;
```

В большинстве реализаций, вы можете установить параметр, называемый **AUTOCOMMIT**. Он будет автоматически запоминать все действия, которые будут выполняться. Действия, которые приведут к ошибке, всегда будут автоматически "прокручены" обратно. Если это предусмотрено в вашей системе, для фиксации всех ваших действий, вы можете использовать эту возможность с помощью команды типа:

```
SET AUTOCOMMIT ON;
```

22. Линейные обыкновенные дифференциальные уравнения и системы. Фундаментальная система решений. Определитель Вронского.

Рассмотрим линейное дифференциальное уравнение n -го порядка

$$a_0(t)y^{(n)}(t) + a_1(t)y^{(n-1)}(t) + \dots + a_{n-1}(t)y'(t) + a_n(t)y(t) = f(t) \quad (3.15)$$

с непрерывными на отрезке $[a, b]$ действительными коэффициентами $a_k(t)$, $k = 0, 1, \dots, n$, $a_0(t) \neq 0$, $t \in [a, b]$ и непрерывной на отрезке $[a, b]$ комплекснозначной функцией $f(t)$.

Введем линейный дифференциальный оператор n -го порядка.

Определение 3.2.1. *Линейным дифференциальным оператором n -го порядка называется оператор*

$$\mathcal{L}y = a_0(t)y^{(n)}(t) + a_1(t)y^{(n-1)}(t) + \dots + a_{n-1}(t)y'(t) + a_n(t)y(t). \quad (3.16)$$

Оператор $\mathcal{L}$ определен для всех n раз непрерывно дифференцируемых на отрезке $[a, b]$ функций $y(t)$, причем $\mathcal{L}y(t) \in C[a, b]$. Используя это определение, уравнение (3.15) можно записать в виде

$$\mathcal{L}y = f(t), \quad t \in [a, b].$$

Если функция $f(t)$ равна нулю на отрезке $[a, b]$, то уравнение (3.15) называется *однородным*, а если функция $f(t)$ не равна нулю на отрезке $[a, b]$, то уравнение (3.15) называется *неоднородным*.

Теорема 3.2.1. *Если функции $y_k(t)$, $k = 1, 2, \dots, m$ являются решениями уравнений $\mathcal{L}y_k = f_k(t)$, то функция $y(t) = \sum_{k=1}^m c_k y_k(t)$, где c_k – комплексные постоянные, является решением уравнения $\mathcal{L}y = f(t)$, где $f(t) = \sum_{k=1}^m c_k f_k(t)$.*

Доказательство. Доказательство этой теоремы следует из линейности оператора $\mathcal{L}$, которая является следствием линейности оператора дифференцирования:

$$\mathcal{L}y = \mathcal{L} \sum_{k=1}^m c_k y_k(t) = \sum_{k=1}^m c_k \mathcal{L}y_k = \sum_{k=1}^m c_k f_k(t) = f(t), \quad t \in [a, b].$$

Следствие 3.2.1. *Линейная комбинация решений однородного уравнения является решением однородного уравнения. Разность двух решений неоднородного уравнения с одинаковой правой частью есть решение однородного уравнения.*

В этом параграфе рассматриваются произвольные скалярные функции

$$\varphi_1(t), \quad \varphi_2(t), \quad \dots, \quad \varphi_m(t),$$

определенные на отрезке $[a, b]$ и принимающие комплексные значения. Никакая связь с решениями дифференциальных уравнений пока не предполагается.

Определение 3.3.1. Скалярные функции $\varphi_1(t), \varphi_2(t), \dots, \varphi_m(t)$ называются линейно зависимыми на отрезке $[a, b]$, если найдутся такие комплексные константы $c_k \in \mathbb{C}$, $k = 1, \dots, m$, $\sum_{k=1}^m |c_k| > 0$, что справедливо равенство

$$c_1\varphi_1(t) + c_2\varphi_2(t) + \dots + c_m\varphi_m(t) = 0, \quad \forall t \in [a, b]. \quad (3.17)$$

Если же равенство (3.17) выполнено только для тривиального набора констант $c_k = 0$, $k = 1, 2, \dots, m$, то скалярные функции $\varphi_1(t), \varphi_2(t), \dots, \varphi_m(t)$ называются линейно независимыми на отрезке $[a, b]$.

Пример 3.3.1. Рассмотрим на отрезке $[a, b]$ функции

$$\varphi_1(t) = t^3, \quad \varphi_2(t) = t^2|t|.$$

Если $0 < a < b$, то на рассматриваемом отрезке $\varphi_1(t) = \varphi_2(t)$ и функции линейно зависимы на $[a, b]$.

Если же $a < 0 < b$, то, положив $t = d = \min\{|a|, b\}$ и $t = -d$ в равенстве $c_1\varphi_1(t) + c_2\varphi_2(t) = 0$, получим систему $c_1d^3 + c_2d^3 = 0$, $c_1d^3 - c_2d^3 = 0$, из которой следует, что $c_1 = c_2 = 0$, а значит $\varphi_1(t) = t^3$ и $\varphi_2(t) = t^2|t|$ линейно независимы на $[a, b]$.

Определение 3.3.2. Определителем Вронского системы функций $\varphi_1(t), \varphi_2(t), \dots, \varphi_m(t)$, состоящей из $(m - 1)$ раз непрерывно дифференцируемых на отрезке $[a, b]$ функций, называется зависящий от переменной $t \in [a, b]$ определитель

$$W[\varphi_1, \dots, \varphi_m](t) = \det \begin{pmatrix} \varphi_1(t) & \varphi_2(t) & \dots & \varphi_m(t) \\ \varphi_1'(t) & \varphi_2'(t) & \dots & \varphi_m'(t) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1^{(m-1)}(t) & \varphi_2^{(m-1)}(t) & \dots & \varphi_m^{(m-1)}(t) \end{pmatrix}.$$

Необходимое условие линейной зависимости скалярных функций устанавливает следующая теорема.

Теорема 3.3.1. Если система $(m-1)$ раз непрерывно дифференцируемых на отрезке $[a, b]$ скалярных функций $\varphi_1(t), \varphi_2(t), \dots, \varphi_m(t)$ является линейно зависимой на отрезке $[a, b]$, то определитель Вронского этой системы тождественно равен нулю на этом отрезке:

$$W[\varphi_1, \dots, \varphi_m](t) = 0, \quad \forall t \in [a, b].$$

Доказательство. Так как функции $\varphi_k(t)$ линейно зависимы на $[a, b]$, то существует нетривиальный набор констант $c_1, c_2, \dots, c_m$, для которого на отрезке $[a, b]$ справедливо равенство (3.17). В этом равенстве допустимо почленное дифференцирование до порядка $m-1$ включительно:

$$c_1 \varphi_1^{(k)}(t) + \dots + c_m \varphi_m^{(k)}(t) = 0, \quad k = 0, 1, \dots, m-1, \quad t \in [a, b]. \quad (3.18)$$

Из (3.18) следует, что вектор-столбцы определителя Вронского линейно зависимы для всех $t \in [a, b]$. Следовательно, этот определитель равен нулю для всех $t \in [a, b]$. $\square$

Следствие 3.3.1. Если для системы $(m-1)$ раз непрерывно дифференцируемых на отрезке $[a, b]$ скалярных функций $\varphi_1(t), \varphi_2(t), \dots, \varphi_m(t)$ определитель Вронского отличен от нуля в некоторой точке $t_0 \in [a, b]$,

$$W[\varphi_1, \dots, \varphi_m](t_0) \neq 0,$$

то эта система является линейно независимой на отрезке $[a, b]$.

Отметим, что равенство нулю определителя Вронского является, вообще говоря, только необходимым условием линейной зависимости скалярных функций. Из равенства нулю определителя Вронского не вытекает их линейная зависимость.

Пример 3.3.2. Для $m = 2$ рассмотрим на отрезке $[-1, 1]$ две функции, имеющие нулевой определитель Вронского:

$$\varphi_1(t) = t^3, \quad \varphi_2(t) = t^2|t|, \quad W[\varphi_1, \varphi_2](t) = \det \begin{pmatrix} t^3 & t^2|t| \\ 3t^2 & 3t|t| \end{pmatrix} \equiv 0.$$

Однако, как показано выше в примере 3.3.1, эти функции являются линейно независимыми на отрезке $[-1, 1]$.

Рассмотрим линейное однородное дифференциальное уравнение порядка n с непрерывными на отрезке $[a, b]$ действительными коэффициентами $a_j(t)$, $j = 0, \dots, n$, $a_0(t) \neq 0$ на $[a, b]$:

$$a_0(t)y^{(n)}(t) + a_1(t)y^{(n-1)}(t) + \dots + a_{n-1}(t)y'(t) + a_n(t)y(t) = 0. \quad (3.19)$$

Рассмотрим систему скалярных функций $y_1(t), y_2(t), \dots, y_n(t)$, являющихся решением линейного однородного уравнения (3.19) порядка n .

Теорема 3.3.2. Для решений $y_1(t), y_2(t), \dots, y_n(t)$ линейного однородного уравнения (3.19) на отрезке $[a, b]$ справедлива следующая альтернатива:

◁ либо $W[y_1, \dots, y_n](t) \equiv 0$ на $[a, b]$ и функции $y_1(t), y_2(t), \dots, y_n(t)$ линейно зависимы на этом отрезке;

◁ либо $W[y_1, \dots, y_n](t) \neq 0 \ \forall t \in [a, b]$ и функции $y_1(t), y_2(t), \dots, y_n(t)$ линейно независимы на $[a, b]$.

Доказательство. Пусть в какой-то точке t_0 определитель Вронского, составленный из функций $y_k(t)$, равен нулю, то есть $W[y_1, \dots, y_n](t_0) = 0$. Рассмотрим систему линейных алгебраических уравнений относительно неизвестных $c_1, c_2, \dots, c_n$:

$$\begin{cases} c_1 y_1(t_0) + c_2 y_2(t_0) + \dots + c_n y_n(t_0) &= 0, \\ c_1 y_1'(t_0) + c_2 y_2'(t_0) + \dots + c_n y_n'(t_0) &= 0, \\ &\dots \\ c_1 y_1^{(n-1)}(t_0) + c_2 y_2^{(n-1)}(t_0) + \dots + c_n y_n^{(n-1)}(t_0) &= 0. \end{cases} \quad (3.20)$$

Так как определитель этой системы равен определителю Вронского и равен нулю ($W[y_1, \dots, y_n](t_0) = 0$), то эта система имеет нетривиальное решение $\tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_n$, $\sum_{k=1}^n |\tilde{c}_k| > 0$.

Рассмотрим функцию

$$\tilde{y}(t) = \sum_{k=1}^n \tilde{c}_k y_k(t).$$

Из теоремы 3.2.1 следует, что эта функция является решением однородного дифференциального уравнения (3.19), а из (3.20) следует, что она удовлетворяет начальным условиям

$$\tilde{y}^{(m)}(t_0) = 0, \quad m = 0, 1, \dots, n-1.$$

Это означает, что функция $\tilde{y}(t)$ является решением однородного дифференциального уравнения (3.19) и удовлетворяет нулевым начальным

условиям в точке t_0 . По теореме единственности решения задачи Коши для линейного дифференциального уравнения эта функция равна нулю на отрезке $[a, b]$. Следовательно,

$$\tilde{y}(t) = \sum_{k=1}^n \tilde{c}_k y_k(t) = 0, \quad t \in [a, b],$$

и функции $y_k(t)$, $k = 1, 2, \dots, n$ линейно зависимы. Тогда из теоремы 3.3.1 следует, что определитель Вронского, составленный из этих функций, равен нулю на отрезке $[a, b]$.

Пусть существует точка $\hat{t} \in [a, b]$ такая, что $W[y_1, \dots, y_n](\hat{t}) \neq 0$. Тогда из предыдущего следует, что определитель Вронского не равен нулю ни в одной точке отрезка $[a, b]$, и функции $y_k(t)$, $k = 1, 2, \dots, n$ линейно независимы на этом отрезке. $\square$

Определение 3.4.1. *Фундаментальной системой решений линейного однородного дифференциального уравнения n -го порядка (3.19) на отрезке $[a, b]$ называется система из n линейно независимых на данном отрезке решений этого уравнения.*

Теорема 3.4.1. *У любого линейного однородного уравнения (3.19) существует фундаментальная система решений на $[a, b]$.*

Доказательство. Рассмотрим постоянную матрицу B с элементами b_{ij} , $i, j = 1, 2, \dots, n$ такую, что $\det B \neq 0$. Обозначим через $y_j(t)$ решения задачи Коши для уравнения (3.19) с начальными условиями

$$y_j(t_0) = b_{1j}, y'_j(t_0) = b_{2j}, \dots, y_j^{(n-1)}(t_0) = b_{nj}, \quad j = 1, 2, \dots, n. \quad (3.21)$$

По теореме 2.3.5 существования и единственности решения задачи Коши для линейного дифференциального уравнения n -го порядка функции $y_j(t)$ существуют и определены однозначно. Составленный из них определитель Вронского $W[y_1, \dots, y_n](t)$, в силу условий (3.21), таков, что $W[y_1, \dots, y_n](t_0) = \det B \neq 0$. Следовательно, по теореме 3.3.2 он не равен нулю ни в одной точке отрезка $[a, b]$, и функции $y_j(t)$ линейно независимы на отрезке $[a, b]$. Значит, они образуют фундаментальную систему решений уравнения (3.19) и теорема доказана. $\square$

Рассмотрим линейное однородное уравнение n -го порядка с вещественными постоянными коэффициентами $a_j \in \mathbb{R}$, $j = 0, \dots, n$, $a_0 \neq 0$:

$$a_0 y^{(n)}(t) + a_1 y^{(n-1)}(t) + \dots + a_{n-1} y'(t) + a_n y(t) = 0. \quad (3.28)$$

Это уравнение можно записать в операторном виде $\mathcal{L}y = 0$, где дифференциальный оператор $\mathcal{L}$ с постоянными коэффициентами

$$\mathcal{L}y = a_0 y^{(n)}(t) + a_1 y^{(n-1)}(t) + \dots + a_{n-1} y'(t) + a_n y(t).$$

Сопоставим дифференциальному оператору $\mathcal{L}$ многочлен

$$M(\lambda) = a_0 \lambda^n + a_1 \lambda^{n-1} + \dots + a_{n-1} \lambda + a_n. \quad (3.29)$$

Многочлен $M(\lambda)$ называется *характеристическим многочленом*, а уравнение

$$M(\lambda) = 0 \quad (3.30)$$

называется *характеристическим уравнением*.

Очевидно, что функция $\exp\{\lambda_0 t\}$ является решением дифференциального уравнения (3.28) тогда и только тогда, когда λ_0 является корнем характеристического уравнения (3.30). Обозначим через $\lambda_1, \dots, \lambda_\ell$ попарно различные корни характеристического многочлена, $M(\lambda_j) = 0$, а через $k_1, \dots, k_\ell$ обозначим кратности этих корней, $k_1 + \dots + k_\ell = n$. Таким образом, справедливо равенство

$$M(\lambda) = a_0(\lambda - \lambda_1)^{k_1}(\lambda - \lambda_2)^{k_2} \dots (\lambda - \lambda_\ell)^{k_\ell}. \quad (3.31)$$

Лемма 3.4.1. *Для любой n раз непрерывно дифференцируемой функции $g(t)$ и произвольного $\lambda \in \mathbb{C}$ справедливо равенство*

$$\mathcal{L}\left(\exp\{\lambda t\}g(t)\right) = \exp\{\lambda t\} \sum_{m=0}^n \frac{M^{(m)}(\lambda)g^{(m)}(t)}{m!}.$$

Доказательство. По формуле Лейбница

$$\begin{aligned} \frac{d^p}{dt^p} \left(\exp\{\lambda t\}g(t) \right) &= \sum_{m=0}^p C_n^p \left(\frac{d^{p-m}}{dt^{p-m}} \exp\{\lambda t\} \right) \left(\frac{d^m}{dt^m} g(t) \right) = \\ &= \exp\{\lambda t\} \sum_{m=0}^p \frac{p(p-1) \dots (p-(m-1))}{m!} \lambda^{p-m} g^{(m)}(t) = \\ &= \exp\{\lambda t\} \sum_{m=0}^p \frac{d^m}{d\lambda^m} \left(\lambda^p \right) \frac{g^{(m)}(t)}{m!}. \end{aligned}$$

Следовательно,

$$\begin{aligned} \mathcal{L}\left(\exp\{\lambda t\}g(t)\right) &= \sum_{p=0}^n a_{n-p} \frac{d^p}{dt^p} \left(\exp\{\lambda t\}g(t) \right) = \\ &= \exp\{\lambda t\} \sum_{p=0}^n a_{n-p} \sum_{m=0}^p \frac{d^m}{d\lambda^m} \left(\lambda^p \right) \frac{g^{(m)}(t)}{m!} \\ &= \exp\{\lambda t\} \sum_{p=0}^n a_{n-p} \sum_{m=0}^n \frac{d^m}{d\lambda^m} \left(\lambda^p \right) \frac{g^{(m)}(t)}{m!}, \end{aligned}$$

так как $d^m \lambda^p / d\lambda^m = 0$, $m = p + 1, \dots, n$. Меняя порядок суммирования, получаем

$$\begin{aligned} \mathcal{L}\left(\exp\{\lambda t\}g(t)\right) &= \exp\{\lambda t\} \sum_{m=0}^n \frac{g^{(m)}(t)}{m!} \frac{d^m}{d\lambda^m} \left(\sum_{p=0}^n a_{n-p} \lambda^p \right) = \\ &= \exp\{\lambda t\} \sum_{m=0}^n \frac{g^{(m)}(t)}{m!} M^{(m)}(\lambda). \end{aligned}$$

□

Лемма 3.4.2. Для каждого корня λ_j характеристического уравнения (3.30) кратности k_j функции

$$\exp\{\lambda_j t\}, \quad t \exp\{\lambda_j t\}, \quad \dots, \quad t^{k_j-1} \exp\{\lambda_j t\}$$

являются решениями однородного уравнения (3.28).

Доказательство. Так как λ_j – корень уравнения (3.30) кратности k_j , то в силу (3.31) справедливо равенство

$$M(\lambda) = (\lambda - \lambda_j)^{k_j} R(\lambda),$$

где $R(\lambda)$ – многочлен степени $n - k_j$. Ясно, что имеют место равенства

$$M^{(m)}(\lambda_j) = \left. \frac{d^m M(\lambda)}{d\lambda^m} \right|_{\lambda=\lambda_j} = 0, \quad m = 0, 1, \dots, k_j - 1.$$

Поэтому из леммы 3.4.1 для $g(t) = t^p$, $p = 0, 1, \dots, k_j - 1$ имеем

$$\begin{aligned} \mathcal{L}\left(\exp\{\lambda_j t\}t^p\right) &= \exp\{\lambda_j t\} \sum_{m=0}^n \frac{(t^p)^{(m)}}{m!} M^{(m)}(\lambda_j) = \\ &= \exp\{\lambda_j t\} \sum_{m=k_j}^n \frac{(t^p)^{(m)}}{m!} M^{(m)}(\lambda_j) = 0 \quad (\text{так как } p < k_j). \end{aligned}$$

□

Таким образом, мы показали, что функции

$$\exp\{\lambda_j t\}, \quad t \exp\{\lambda_j t\}, \quad \dots, \quad t^{k_j-1} \exp\{\lambda_j t\}, \quad j = 1, \dots, \ell. \quad (3.32)$$

Теорема 3.4.4. Система функций (3.32) составляет фундаментальную систему решений линейного однородного уравнения с постоянными коэффициентами (3.28) на любом отрезке $[a, b]$.

Доказательство. Для доказательства теоремы достаточно доказать, что система функций (3.32) является линейно независимой на любом отрезке $[a, b]$. Предположим, что нетривиальная линейная комбинация функций из системы (3.32) обращается тождественно в ноль на некотором отрезке:

$$\sum_{k=0}^{k_1-1} C_{1,k} t^k \exp\{\lambda_1 t\} + \sum_{k=0}^{k_2-1} C_{2,k} t^k \exp\{\lambda_2 t\} + \cdots + \sum_{k=0}^{k_\ell-1} C_{\ell,k} t^k \exp\{\lambda_\ell t\} \equiv 0,$$

или

$$P_1(t) \exp\{\lambda_1 t\} + P_2(t) \exp\{\lambda_2 t\} + \cdots + P_\ell(t) \exp\{\lambda_\ell t\} \equiv 0, \quad (3.33)$$

где степень многочлена $s_j = \deg P_j(t) \leq k_j - 1$, $j = 1, \dots, \ell$. Без ограничения общности можно считать, что многочлен $P_\ell(t)$ нетривиален, $P_\ell(t) = p_\ell t^{s_\ell} + \dots$, $p_\ell \neq 0$. После умножения (3.33) на $\exp\{-\lambda_1 t\}$ получаем

$$P_1(t) + P_2(t) \exp\{(\lambda_2 - \lambda_1)t\} + \cdots + P_\ell(t) \exp\{(\lambda_\ell - \lambda_1)t\} \equiv 0.$$

Теорема 3.4.4. Система функций (3.32) составляет фундаментальную систему решений линейного однородного уравнения с постоянными коэффициентами (3.28) на любом отрезке $[a, b]$.

Доказательство. Для доказательства теоремы достаточно доказать, что система функций (3.32) является линейно независимой на любом отрезке $[a, b]$. Предположим, что нетривиальная линейная комбинация функций из системы (3.32) обращается тождественно в ноль на некотором отрезке:

$$\sum_{k=0}^{k_1-1} C_{1,k} t^k \exp\{\lambda_1 t\} + \sum_{k=0}^{k_2-1} C_{2,k} t^k \exp\{\lambda_2 t\} + \cdots + \sum_{k=0}^{k_\ell-1} C_{\ell,k} t^k \exp\{\lambda_\ell t\} \equiv 0,$$

или

$$P_1(t) \exp\{\lambda_1 t\} + P_2(t) \exp\{\lambda_2 t\} + \cdots + P_\ell(t) \exp\{\lambda_\ell t\} \equiv 0, \quad (3.33)$$

где степень многочлена $s_j = \deg P_j(t) \leq k_j - 1$, $j = 1, \dots, \ell$. Без ограничения общности можно считать, что многочлен $P_\ell(t)$ нетривиален, $P_\ell(t) = p_\ell t^{s_\ell} + \dots$, $p_\ell \neq 0$. После умножения (3.33) на $\exp\{-\lambda_1 t\}$ получаем

$$P_1(t) + P_2(t) \exp\{(\lambda_2 - \lambda_1)t\} + \cdots + P_\ell(t) \exp\{(\lambda_\ell - \lambda_1)t\} \equiv 0.$$

Так как все коэффициенты уравнения (3.28) вещественны, то фундаментальную систему решений можно также конструктивно построить

в вещественном виде. Характеристический многочлен в (3.29) имеет вещественные коэффициенты. Как следует из курса линейной алгебры, его комплекснозначные корни идут комплексно сопряженными парами: $\lambda = \alpha + i\beta$, $\lambda^* = \alpha - i\beta$, $\alpha, \beta \in \mathbb{R}$. Тогда в построенной фундаментальной системе решений (3.32) функции, отвечающие вещественным корням характеристического многочлена $M_n(\Lambda)$, являются вещественными, а отвечающие комплексным корням функции встречаются только комплексно сопряженными парами:

$$y(t) = t^s \exp\{\alpha t\}(\cos \beta t + i \sin \beta t), \quad y^*(t) = t^s \exp\{\alpha t\}(\cos \beta t - i \sin \beta t).$$

Заменим каждую пару таких функций соответствующими действительными и мнимыми частями:

$$\begin{aligned} y_R(t) &= \operatorname{Re} y(t) = t^s \exp\{\alpha t\} \cos \beta t, \\ y_I(t) &= \operatorname{Im} y(t) = t^s \exp\{\alpha t\} \sin \beta t. \end{aligned} \quad (3.34)$$

$$\Delta'(t) = \det \begin{pmatrix} y_1(t) & y_2(t) & \dots & y_{n-1}(t) & y_n(t) \\ y_1'(t) & y_2'(t) & \dots & y_{n-1}'(t) & y_n'(t) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ y_1^{(n-2)}(t) & y_2^{(n-2)}(t) & \dots & y_{n-1}^{(n-2)}(t) & y_n^{(n-2)}(t) \\ y_1^{(n)}(t) & y_2^{(n)}(t) & \dots & y_{n-1}^{(n)}(t) & y_n^{(n)}(t) \end{pmatrix}.$$

$$a_1(t) = -\frac{\Delta'(t)}{\Delta(t)}.$$

Интегрируя от t_0 до t , получим формулу Остроградского-Лиувилля

$$\Delta(t) = \Delta(t_0) \exp\left\{-\int_{t_0}^t a_1(\tau) d\tau\right\}, \quad t \in [a, b].$$

Следствие 3.5.1. Если коэффициент $a_1(t) = 0$, $t \in [a, b]$, то определитель Вронского $W[y_1, y_2, \dots, y_n](t)$ постоянен на отрезке $[a, b]$.

Рассмотрим на отрезке $[a, b]$ нормальную систему линейных обыкновенных дифференциальных уравнений первого порядка в векторной форме с непрерывными на отрезке $[a, b]$ действительными коэффициентами $a_{i,j}(t)$ и непрерывными комплекснозначными $f_k(t)$:

$$\frac{d\bar{y}(t)}{dt} = A(t)\bar{y}(t) + \bar{f}(t), \quad t \in [a, b], \quad (4.1)$$

где

$$A(t) = \begin{pmatrix} a_{11}(t) & \cdots & a_{1n}(t) \\ \vdots & \ddots & \vdots \\ a_{n1}(t) & \cdots & a_{nn}(t) \end{pmatrix}, \quad \bar{f}(t) = \begin{pmatrix} f_1(t) \\ \vdots \\ f_n(t) \end{pmatrix}.$$

Напомним, что решение $\bar{y}(t) = (y_1(t), \dots, y_n(t))^T$ системы (4.1) является, вообще говоря, комплекснозначной вектор-функцией $\bar{y}(t) = \bar{u}(t) + i\bar{v}(t)$, где

$$\bar{u}(t) = (u_1(t), \dots, u_n(t))^T, \quad \bar{v}(t) = (v_1(t), \dots, v_n(t))^T,$$

а $u_j(t), v_j(t)$ действительны, $j = 1, \dots, n$. В дальнейшем, если не оговорено особо, речь пойдет именно о комплекснозначных решениях.

Определение 4.1.1. Система (4.1) называется однородной, если $\bar{f}(t) \equiv \bar{\theta}$ на отрезке $[a, b]$. В противном случае система (4.1) называется неоднородной.

Лемма 4.1.1. Если $\bar{y}(t)$ – решение линейной однородной системы дифференциальных уравнений, то $\alpha\bar{y}(t)$ также решение однородной системы для любого $\alpha \in \mathbb{C}$. Если $\bar{y}_1(t)$ и $\bar{y}_2(t)$ – два решения линейной однородной системы, то $\bar{y}(t) = \bar{y}_1(t) + \bar{y}_2(t)$ также решение однородной системы.

Доказательство. Если $d\bar{y}(t)/dt = A(t)\bar{y}(t)$, то

$$\frac{d\{\alpha\bar{y}(t)\}}{dt} = \alpha \frac{d\bar{y}(t)}{dt} = \alpha A(t)\bar{y}(t) = A(t)\{\alpha\bar{y}(t)\}.$$

Если $d\bar{y}_\ell(t)/dt = A(t)\bar{y}_\ell(t)$, $\ell = 1, 2$, то

$$\begin{aligned} \frac{d\bar{y}(t)}{dt} &= \frac{d\{\bar{y}_1(t) + \bar{y}_2(t)\}}{dt} = \frac{d\bar{y}_1(t)}{dt} + \frac{d\bar{y}_2(t)}{dt} = \\ &= A(t)\bar{y}_1(t) + A(t)\bar{y}_2(t) = A(t)\bar{y}(t). \end{aligned}$$

Следствие 4.1.1. Если $\bar{y}_\ell(t)$ – решения линейной однородной системы $\ell = 1, \dots, m$, то $\bar{y}(t) = \sum_{\ell=1}^m \alpha_\ell \bar{y}_\ell(t)$ также решение однородной системы для любых $\alpha_\ell \in \mathbb{C}$.

Рассмотрим линейную однородную систему обыкновенных дифференциальных уравнений с непрерывными на отрезке $[a, b]$ действительными коэффициентами $a_{i,j}(t)$, $i, j = 1, 2, \dots, n$

$$\frac{d\bar{y}(t)}{dt} = A(t)\bar{y}(t), \quad t \in [a, b], \quad (4.2)$$

где

$$A(t) = \begin{pmatrix} a_{11}(t) & \cdots & a_{1n}(t) \\ \vdots & \ddots & \vdots \\ a_{n1}(t) & \cdots & a_{nn}(t) \end{pmatrix}, \quad \bar{y}(t) = \begin{pmatrix} y_1(t) \\ \vdots \\ y_n(t) \end{pmatrix}.$$

Пусть имеется n вектор-функций

$$\bar{y}_j(t) = (y_{1j}(t), \dots, y_{nj}(t))^T, \quad j = 1, \dots, n.$$

Составим матрицу $Y(t)$, столбцами которой являются данные вектор-функции:

$$Y(t) = (\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_n(t)) = \begin{pmatrix} y_{11}(t) & \cdots & y_{1n}(t) \\ \vdots & \ddots & \vdots \\ y_{n1}(t) & \cdots & y_{nn}(t) \end{pmatrix}. \quad (4.3)$$

Сопоставим системе (4.2) матричное однородное дифференциальное уравнение

$$\frac{dY(t)}{dt} = A(t)Y(t), \quad (4.4)$$

Теорема 4.1.1. Вектор-функции $\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_n(t)$ являются решениями однородной системы (4.2) на отрезке $[a, b]$ тогда и только тогда, когда составленная из этих функций матрица $Y(t)$ вида (4.3) является решением матричного дифференциального уравнения (4.4).

Доказательство. Для доказательства необходимости рассмотрим решения $\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_n(t)$ системы (4.2) и составим из них матрицу $Y(t)$ вида (4.3). Поскольку

$$\frac{d\bar{y}_j(t)}{dt} = A(t)\bar{y}_j(t), \quad j = 1, \dots, n,$$

то для соответствующей матричной производной, элементы которой сгруппированы по столбцам, получаем равенства

$$\begin{aligned} \frac{dY(t)}{dt} &= \left(\frac{d\bar{y}_1(t)}{dt}, \frac{d\bar{y}_2(t)}{dt}, \dots, \frac{d\bar{y}_n(t)}{dt} \right) = \\ &= (A\bar{y}_1(t), A\bar{y}_2(t), \dots, A\bar{y}_n(t)) = A(t)Y(t). \end{aligned}$$

То есть выполнено матричное уравнение (4.4). Аналогично, расписывая матричное уравнение (4.4) по столбцам, доказывается достаточность. $\square$

Теорема 4.1.2. Пусть матричная функция $Y(t)$ является решением матричного уравнения (4.4). Тогда:

1. для любого вектора констант $\bar{c} = (c_1, c_2, \dots, c_n)^\top$, $c_j \in \mathbb{C}$, вектор-функция $\bar{y}(t) = Y(t)\bar{c}$ удовлетворяет системе (4.2);
2. для любой матрицы констант $B = (b_{i,j})$, $b_{i,j} \in \mathbb{C}$, $i, j = 1, \dots, n$, матричная функция $X(t) = Y(t)B$ удовлетворяет уравнению (4.4).

Доказательство. 1. Если матричная функция

$$Y(t) = (\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_n(t))$$

является решением уравнения (4.4), то по теореме 4.1.1 вектор-столбцы $\bar{y}_j(t)$ являются решениями системы (4.2), также как и их линейная комбинация

$$\bar{y}(t) = Y(t)\bar{c} = \sum_{j=1}^n c_j \bar{y}_j(t).$$

2. В силу линейности операции дифференцирования и ассоциативности операции произведения матриц, имеем:

$$\begin{aligned}\frac{dX(t)}{dt} &= \frac{d}{dt} \{Y(t)B\} = \frac{dY(t)}{dt} \cdot B = \\ &= \{A(t)Y(t)\} B = A(t) \{Y(t)B\} = A(t)X(t).\end{aligned}$$

□

Определение 4.2.1. Вектор-функции $\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_m(t)$ называются линейно зависимыми на отрезке $[a, b]$, если найдутся комплексные константы $c_1, c_2, \dots, c_m$, $\sum_{j=1}^m |c_j| > 0$ такие, что

$$c_1 \bar{y}_1(t) + c_2 \bar{y}_2(t) + \dots + c_m \bar{y}_m(t) = \bar{\theta}, \quad \forall t \in [a, b]. \quad (4.5)$$

Если же равенство (4.5) выполнено только для тривиального вектора констант, $\bar{c} = (0, \dots, 0)^\top$, то вектор-функции $\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_m(t)$ называются линейно независимыми на отрезке $[a, b]$.

Определение 4.2.2. Определителем Вронского системы заданных на отрезке $[a, b]$ вектор функций $\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_m(t)$ называется зависящий от переменной $t \in [a, b]$ определитель матричной функции $Y(t) = (\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_m(t))$:

$$\Delta(t) = \det Y(t).$$

Теорема 4.2.1. Если система вектор функций $\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_m(t)$ является линейно зависимой на отрезке $[a, b]$, то определитель Вронского этой системы тождественно равен нулю на этом отрезке:

$$\Delta(t) = 0, \quad \forall t \in [a, b].$$

Доказательство. Из условия линейной зависимости (4.6) вытекает существование такого ненулевого вектора $\bar{c} = (c_1, \dots, c_m)^\top$, что для произвольного фиксированного $t_0 \in [a, b]$ справедливо равенство

$$Y(t_0)\bar{c} = \bar{\theta}. \quad (4.7)$$

Равенство (4.7) означает, что однородная система линейных алгебраических уравнений с числовой матрицей $Y(t_0)$ имеет нетривиальное решение $\bar{c}$. По известной теореме алгебры это возможно только для вырожденной матрицы, то есть $\det Y(t_0) = 0$. □

Теорема 4.2.2. Пусть $\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_n(t)$ – система вектор-функций решений линейной однородной системы (4.2) на отрезке $[a, b]$. Если найдется точка $t_0 \in [a, b]$, для которой

$$\det Y(t_0) = 0,$$

то система вектор-функций $\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_n(t)$ линейно зависима на отрезке $[a, b]$ и

$$\det Y(t) = 0, \quad \forall t \in [a, b].$$

Доказательство. Однородная система линейных алгебраических уравнений относительно вектора $\bar{c} = (c_1, \dots, c_n)^\top$

$$Y(t_0)\bar{c} = \bar{\theta} \tag{4.8}$$

имеет ненулевое решение $\bar{c}^0 = (c_1^0, \dots, c_n^0)^\top$ в силу вырожденности числовой матрицы $Y(t_0)$, имеющей нулевой определитель.

Положим $\bar{y}(t) = Y(t)\bar{c}^0$. Ясно, что $\bar{y}(t)$ – решение однородной системы (4.2) в силу первой части теоремы 4.1.2 и, кроме того, $\bar{y}(t_0) = \bar{\theta}$ в силу (4.8). Таким образом, построенная функция является решением задачи Коши с нулевым начальным условием при $t = t_0$:

$$\frac{d\bar{y}(t)}{dt} = A(t)\bar{y}(t), \quad \bar{y}(t_0) = \bar{\theta}.$$

Эта задача Коши по теореме существования и единственности 2.1.2 имеет на рассматриваемом отрезке только одно решение – нулевое. Поэтому

$$\bar{\theta} = \bar{y}(t) = Y(t)\bar{c}^0 = c_1^0\bar{y}_1(t) + c_2^0\bar{y}_2(t) + \dots + c_n^0\bar{y}_n(t), \quad \forall t \in [a, b],$$

и рассматриваемая система вектор-функций является линейно зависимой на отрезке $[a, b]$. Тогда в силу теоремы 4.2.1 имеем $\det Y(t) = 0$, $\forall t \in [a, b]$. $\square$

Теорема 4.2.3. Определитель Вронского для вектор-функций $\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_n(t)$, являющихся решениями линейной однородной системы дифференциальных уравнений (4.2) на отрезке $[a, b]$, либо тождественно равен нулю, $\det Y(t) \equiv 0$ (и система вектор-функций линейно зависима), либо не обращается в ноль ни в одной точке, $\det Y(t) \neq 0$, $\forall t \in [a, b]$ (и система вектор-функций линейно независима).

Определение 4.3.1. Фундаментальной системой решений линейной однородной системы дифференциальных уравнений $\frac{d\bar{y}(t)}{dt} = A(t)\bar{y}(t)$ порядка n на отрезке $[a, b]$ называется совокупность n линейно независимых решений $\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_n(t)$ этой системы. Соответствующая этим решениям функциональная матрица

$$Y(t) = (\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_n(t))$$

называется *фундаментальной матрицей*.

Теорема 4.3.1. Для любой однородной системы линейных дифференциальных уравнений вида (4.2) с непрерывными на отрезке $[a, b]$ коэффициентами существует фундаментальная система решений.

Доказательство. Зафиксируем любое $t_0 \in [a, b]$ и рассмотрим задачу Коши для матричного дифференциального уравнения

$$\frac{dY(t)}{dt} = A(t)Y(t), \quad Y(t_0) = E, \quad (4.9)$$

где E – единичная матрица. Расписывая матричные равенства по столбцам, заключаем, что задача (4.9) эквивалентна совокупности из n задач Коши

$$\frac{d\bar{y}_j(t)}{dt} = A(t)\bar{y}_j(t), \quad \bar{y}_j(t_0) = (0, \dots, 0, \underbrace{1}_j, 0, \dots, 0)^\top, \quad j = 1, \dots, n,$$

отличающихся лишь начальными данными. Существование на всем отрезке $[a, b]$ решений $\bar{y}_j(t)$ этих задач Коши, а значит и решения $Y(t)$ матричной задачи (4.9), вытекает из теоремы 2.1.2. Поскольку определитель матричной функции $Y(t)$ в силу (4.9) равен 1, $\det Y(t_0) = \det E = 1$, то линейная независимость на рассматриваемом отрезке построенной системы решений $\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_n(t)$ есть следствие теоремы 4.2.3 об альтернативе для определителя Вронского. Таким образом, $\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_n(t)$ – фундаментальная система решений, а $Y(t)$ – фундаментальная матрица. $\square$

Определение 4.3.2. Общим решением линейной однородной системы дифференциальных уравнений n -го порядка называется зависящее от n произвольных постоянных решение этого уравнения такое, что любое другое решение системы может быть получено из него в результате выбора некоторых значений этих постоянных.

Теорема 4.3.2. Пусть $Y(t) = (\bar{y}_1(t), \bar{y}_2(t), \dots, \bar{y}_n(t))$ – фундаментальная матрица для линейной однородной системы

$$\frac{d\bar{y}(t)}{dt} = A(t)\bar{y}(t)$$

на отрезке $[a, b]$. Тогда ее общее решение представимо в виде

$$\bar{y}_{OO}(t) = c_1\bar{y}_1(t) + c_2\bar{y}_2(t) + \dots + c_n\bar{y}_n(t) = Y(t)\bar{c}, \quad (4.10)$$

где $c_1, c_2, \dots, c_n$ – произвольные постоянные, $\bar{c} = (c_1, c_2, \dots, c_n)$.

Доказательство. По теореме 4.1.2 вектор-функция $Y(t)\bar{c}$ является решением однородной системы для любых $\bar{c} \in \mathbb{C}^n$. Согласно определению общего решения осталось показать, что для любого наперед заданного решения $\bar{y}(t)$ линейной однородной системы найдется вектор констант $\tilde{c} \in \mathbb{C}^n$ такой, что на отрезке $[a, b]$ выполнено равенство

$$\bar{y}(t) = Y(t)\tilde{c}. \quad (4.11)$$

Для построения $\tilde{c}$ зафиксируем произвольное $t_0 \in [a, b]$ и вычислим $\bar{y}^0 = \bar{y}(t_0)$. Рассмотрим систему линейных алгебраических уравнений относительно $\tilde{c} = (\tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_n)^\top$:

$$Y(t_0)\tilde{c} = \bar{y}^0. \quad (4.12)$$

В силу невырожденности матрицы $Y(t_0)$ с определителем $\det Y(t_0) \neq 0$ эта система имеет единственное решение $\tilde{c} = (\tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_n)^\top$. Тогда функции $\tilde{y}(t) = Y(t)\tilde{c}$ и $\bar{y}(t)$ являются решениями одной и той же задачи Коши

$$\frac{d\bar{y}(t)}{dt} = A(t)\bar{y}(t), \quad \bar{y}(t_0) = \bar{y}^0, \quad (4.13)$$

и по теореме единственности обязаны совпадать, что доказывает (4.11). Отметим, что для фиксированного решения $\bar{y}(t)$ вектор констант $\tilde{c} \in \mathbb{C}^n$ в представлении (4.11) определен однозначно. $\square$

Теорема 4.3.3. *Общее решение $\bar{y}_{OH}(t)$ линейной неоднородной системы дифференциальных уравнений (4.15) представимо в виде*

$$\bar{y}_{OH}(t) = Y(t)\bar{c} + \bar{y}_H(t), \quad \forall \bar{c} = (c_1, c_2, \dots, c_n)^\top \in \mathbb{C}^n, \quad (4.16)$$

где $\bar{y}_H(t)$ – некоторое (частное) решение неоднородной системы (4.15).

Рассмотрим однородную систему обыкновенных дифференциальных уравнений с постоянной матрицей коэффициентов $A(t) \equiv A = (a_{i,j})$, $a_{i,j} \in \mathbb{R}$, $i, j = 1, \dots, n$:

$$\frac{d\bar{y}(t)}{dt} = A\bar{y}(t). \quad (4.23)$$

По аналогии со скалярным уравнением $y'(t) = ay(t)$, которое имеет решение $y(t) = h \exp\{at\}$ для любого $h \in \mathbb{C}$, будем искать нетривиальные решения системы (4.23) в виде

$$\bar{y}(t) = \bar{h} \exp\{\lambda t\}, \quad \bar{h} = (h_1, h_2, \dots, h_n)^\top \in \mathbb{C}^n, \quad \lambda \in \mathbb{C}. \quad (4.24)$$

Подстановка вектор-функции (4.24) в систему (4.23) приводит к задаче нахождения таких $\lambda \in \mathbb{C}$, при которых система линейных алгебраических уравнений

$$(A - \lambda E)\bar{h} = \bar{\theta} \quad (4.25)$$

имеет нетривиальное решение $\bar{h}$. Как известно из курса линейной алгебры, такие λ называются *собственными значениями* матрицы A , а отвечающие им векторы $\bar{h}$ — *собственными векторами* матрицы A . Собственные значения и только они являются корнями *характеристического многочлена* $M(\lambda)$:

$$M(\lambda) = \det(A - \lambda E) = 0. \quad (4.26)$$

Теорема 4.4.1. Пусть у матрицы A имеется ровно n линейно независимых собственных векторов

$$\bar{h}_1, \quad \bar{h}_2, \quad \dots, \quad \bar{h}_n,$$

отвечающих соответствующим собственным значениям

$$\lambda_1, \quad \lambda_2, \quad \dots, \quad \lambda_n.$$

Тогда вектор-функции

$$\bar{y}_1(t) = \bar{h}_1 \exp\{\lambda_1 t\}, \quad \bar{y}_2(t) = \bar{h}_2 \exp\{\lambda_2 t\}, \quad \dots \quad \bar{y}_n(t) = \bar{h}_n \exp\{\lambda_n t\} \quad (4.27)$$

образуют фундаментальную систему решений (4.23) на произвольном отрезке $[a, b]$.

4.4.2. Построение фундаментальной системы решений, когда не существует базиса из собственных векторов

Рассмотрим случай, когда количество существующих у матрицы A линейно независимых собственных векторов строго меньше, чем порядок системы n . Выпишем все попарно различные собственные значения λ_j с соответствующими кратностями k_j :

$$\begin{aligned} \lambda_1, \quad \lambda_2, \quad \dots, \quad \lambda_\ell, \quad \lambda_i \neq \lambda_j \text{ при } i \neq j, \\ k_1, \quad k_2, \quad \dots, \quad k_\ell, \quad k_j \geq 1, \quad k_1 + k_2 + \dots + k_\ell = n. \end{aligned}$$

Пусть далее $\lambda \in \{\lambda_1, \dots, \lambda_\ell\}$ обозначает одно из собственных значений с соответствующей кратностью k . Покажем, что каждому такому собственному значению можно сопоставить ровно k вектор-функций, являющихся решениями однородной системы (4.23). Если размерность $s = \dim \text{Ker}(A - \lambda E)$ собственного подпространства, определяющая количество линейно независимых собственных векторов для данного собственного значения, равна кратности собственного значения, $s = k$, то искомые функции строятся согласно (4.27).

Если размерность собственного подпространства меньше кратности собственного значения, $s < k$, то, как известно из курса линейной алгебры, можно выбрать собственные векторы $\bar{h}_1^1, \bar{h}_2^1, \dots, \bar{h}_s^1$ так, что состоящая ровно из k векторов система собственных векторов $\bar{h}_j^1$ и присоединенных векторов $\bar{h}_j^m$, $m = 2, \dots, p_j$, $j = 1, \dots, s$, $p_j \geq 1$, $p_1 + p_2 + \dots + p_s = k$, которую запишем в виде

$$\begin{array}{ccccc} \bar{h}_1^1, & \dots & \bar{h}_j^1, & \dots & \bar{h}_s^1, \\ \bar{h}_1^2, & \dots & \bar{h}_j^2, & \dots & \bar{h}_s^2, \\ \vdots & \dots & \vdots & \dots & \vdots \\ \bar{h}_1^{p_1}, & \dots & \bar{h}_j^{p_j}, & \dots & \bar{h}_s^{p_s}, \end{array}$$

удовлетворяет уравнениям

$$\begin{aligned} A\bar{h}_j^1 &= \lambda\bar{h}_j^1, \\ A\bar{h}_j^2 &= \lambda\bar{h}_j^2 + \bar{h}_j^1, \\ &\dots \\ A\bar{h}_j^m &= \lambda\bar{h}_j^m + \bar{h}_j^{m-1}, \\ &\dots \\ A\bar{h}_j^{p_j} &= \lambda\bar{h}_j^{p_j} + \bar{h}_j^{p_j-1}. \end{aligned} \tag{4.28}$$

С помощью собственных и присоединенных векторов построим семейство из следующих k функций

$$\begin{aligned} \bar{y}_j^1(t) &= \bar{h}_j^1 \exp\{\lambda t\}, \\ \bar{y}_j^2(t) &= \left(\bar{h}_j^2 + \frac{t}{1!} \bar{h}_j^1 \right) \exp\{\lambda t\}, \\ &\vdots \end{aligned} \tag{4.29}$$

$$\begin{aligned}
\bar{y}_j^m(t) &= \left(\bar{h}_j^m + \frac{t}{1!} \bar{h}_j^{m-1} + \frac{t^2}{2!} \bar{h}_j^{m-2} + \dots + \frac{t^q}{q!} \bar{h}_j^{m-q} + \dots \right. \\
&\quad \left. \dots + \frac{t^{m-1}}{(m-1)!} \bar{h}_j^1 \right) \exp\{\lambda t\}, \\
&\quad \vdots \\
\bar{y}_j^{p_j}(t) &= \left(\bar{h}_j^{p_j} + \frac{t}{1!} \bar{h}_j^{p_j-1} + \frac{t^2}{2!} \bar{h}_j^{p_j-2} + \dots + \frac{t^q}{q!} \bar{h}_j^{p_j-q} + \dots \right. \\
&\quad \left. \dots + \frac{t^{p_j-1}}{(p_j-1)!} \bar{h}_j^1 \right) \exp\{\lambda t\}, \\
&\quad j = 1, \dots, s.
\end{aligned}$$

Докажем, что все функции из построенного семейства являются решениями линейной однородной системы (4.23). Рассмотрим функцию $\bar{y}_j^m(t)$, вычислим ее производную $d\bar{y}_j^m(t)/dt$ и сгруппируем результат так, чтобы удобно было воспользоваться соотношениями (4.28). Имеем

$$\begin{aligned}
\frac{d\bar{y}_j^m(t)}{dt} &= \\
&= \left(\bar{h}_j^{m-1} + \frac{t}{1!} \bar{h}_j^{m-2} + \frac{t^2}{2!} \bar{h}_j^{m-3} + \dots + \frac{t^q}{q!} \bar{h}_j^{m-q-1} + \dots + \frac{t^{m-2}}{(m-2)!} \bar{h}_j^1 + \right. \\
&\quad + \lambda \bar{h}_j^m + \frac{t}{1!} \lambda \bar{h}_j^{m-1} + \frac{t^2}{2!} \lambda \bar{h}_j^{m-2} + \dots + \frac{t^q}{q!} \lambda \bar{h}_j^{m-q} + \dots + \frac{t^{m-2}}{(m-2)!} \lambda \bar{h}_j^2 + \\
&\quad \left. + \frac{t^{m-1}}{(m-1)!} \lambda \bar{h}_j^1 \right) \exp\{\lambda t\} = \\
&= \left(A \bar{h}_j^m + \frac{t}{1!} A \bar{h}_j^{m-1} + \frac{t^2}{2!} A \bar{h}_j^{m-2} + \dots + \frac{t^q}{q!} A \bar{h}_j^{m-q} + \dots \right. \\
&\quad \left. \dots + \frac{t^{m-1}}{(m-1)!} A \bar{h}_j^1 \right) \exp\{\lambda t\} = A \bar{y}_j^m(t),
\end{aligned}$$

$$m = 1, \dots, p_j, \quad j = 1, \dots, s.$$

Следовательно, $\bar{y}_j^m(t)$ – решения системы (4.23).

Докажем, что система из n вектор-функций, состоящая из объединения построенных для всех $\lambda \in \{\lambda_1, \dots, \lambda_\ell\}$ решений вида (4.29), является линейно независимой на произвольном отрезке $[a, b]$. Рассмотрим отрезок $[c, d]$, $[a, b] \subseteq [c, d]$, $0 \in [c, d]$. Вектор-функции из (4.29)

являются решениями системы (4.23) на отрезке $[c, d]$. В принадлежащей этому отрезку точке $t = 0$ определитель Вронского этой системы отличен от нуля, поскольку соответствующая матрица $Y(0)$ составлена из столбцов, являющихся собственными и присоединенными векторами матрицы A , совокупность которых линейно независима и образует базис в $\mathbb{C}^n$. Согласно теореме 4.2.3 об альтернативе для определителя Вронского, $\det Y(t) \neq 0$ на всем отрезке $[c, d]$, а значит и на его части $[a, b]$. Поэтому рассматриваемая система решений (4.23) является линейно независимой на $[a, b]$ и, следовательно, составляет фундаментальную систему решений на этом отрезке. Тем самым установлена справедливость следующей теоремы.

Теорема 4.4.2. Система из n вектор-функций, состоящая из объединения построенных для всех различных собственных значений $\lambda_1, \lambda_2, \dots, \lambda_\ell$ решений вида (4.29), является фундаментальной системой решений (4.23) на произвольном отрезке $[a, b]$.

матрицы системы) идут комплексно сопряженными парами: $\lambda = p + iq$, $\lambda^* = p - iq$, $M(\lambda) = 0$, $M(\lambda^*) = 0$. Тогда в построенной в теореме 4.4.2 фундаментальной системе решений вектор-функции, отвечающие вещественным собственным значениям, являются вещественными, а отвечающие комплексным собственным значениям функции встречаются только комплексно сопряженными парами. Заменим в фундаментальной системе решений каждую такую пару функций

$$\bar{y}(t) = (y_1(t), \dots, y_n(t))^T, \quad \bar{y}^*(t) = (y_1^*(t), \dots, y_n^*(t))^T$$

соответствующими действительными и мнимыми частями,

$$\bar{y}^R(t) = \operatorname{Re} \bar{y}(t), \quad \bar{y}^I(t) = \operatorname{Im} \bar{y}(t).$$

Так как

$$\bar{y}^R(t) = 0.5(\bar{y}(t) + \bar{y}^*(t)), \quad \bar{y}^I(t) = 0.5i(\bar{y}^*(t) - \bar{y}(t)), \quad (4.30)$$

то $\bar{y}^R(t), \bar{y}^I(t)$ – решения однородной системы как линейные комбинации решений. Построенная таким образом совокупность вектор-функций состоит из n вещественных решений линейной однородной системы дифференциальных уравнений и задает ее фундаментальную систему решений.

24. Теоремы существования и единственности решения задачи Коши для обыкновенного дифференциального уравнения первого порядка, разрешенного относительно производной.

Пусть функция $f(t, y)$ определена и непрерывна в прямоугольнике

$$\Pi = \{(t, y) : |t - t_0| \leq T, |y - y_0| \leq A\}.$$

Рассмотрим на отрезке $[t_0 - T, t_0 + T]$ дифференциальное уравнение

$$y'(t) = f(t, y(t)) \quad (2.1)$$

с условием

$$y(t_0) = y_0. \quad (2.2)$$

Требуется определить функцию $y(t)$, удовлетворяющую уравнению (2.1) и условию (2.2). Эта задача называется задачей с начальным условием или задачей Коши.

Рассмотрим отрезок $[t_1, t_2]$ такой, что $t_0 - T \leq t_1 < t_2 \leq t_0 + T$, $t_0 \in [t_1, t_2]$.

Определение 2.1.1. Функция $\bar{y}(t)$ называется решением задачи Коши (2.1), (2.2) на отрезке $[t_1, t_2]$, если: $\bar{y}(t) \in C^1[t_1, t_2]$, $|\bar{y}(t) - y_0| \leq A$ для $t \in [t_1, t_2]$, $\bar{y}(t)$ удовлетворяет уравнению (2.1) для $t \in [t_1, t_2]$ и условию (2.2).

Покажем, что решение задачи с начальным условием (2.1), (2.2) эквивалентно решению некоторого интегрального уравнения.

Рассмотрим на отрезке $[t_0 - T, t_0 + T]$ уравнение относительно неизвестной функции $y(t)$

$$y(t) = y_0 + \int_{t_0}^t f(\tau, y(\tau)) d\tau. \quad (2.3)$$

Такое уравнение называется интегральным, поскольку неизвестная функция $y(t)$ входит под знак интеграла.

Лемма 2.1.1. Функция $\bar{y}(t)$ является решением задачи Коши (2.1), (2.2) на отрезке $[t_1, t_2]$ тогда и только тогда, когда $\bar{y}(t) \in C[t_1, t_2]$, $|\bar{y}(t) - y_0| \leq A$ для $t \in [t_1, t_2]$ и $\bar{y}(t)$ удовлетворяет уравнению (2.3) для $t \in [t_1, t_2]$.

Доказательство. Пусть функция $\bar{y}(t)$ является решением задачи с начальным условием (2.1), (2.2) на отрезке $[t_1, t_2]$. Из определения 2.1.1 следует, что $\bar{y}(t) \in C[t_1, t_2]$, $|\bar{y}(t) - y_0| \leq A$ для $t \in [t_1, t_2]$. Покажем, что $\bar{y}(t)$ удовлетворяет уравнению (2.3) для $t \in [t_1, t_2]$. Интегрируя уравнение (2.1) от t_0 до t , получим

$$\int_{t_0}^t \bar{y}'(\tau) d\tau = \int_{t_0}^t f(\tau, \bar{y}(\tau)) d\tau, \quad t \in [t_1, t_2].$$

Учитывая начальное условие (2.2), имеем

$$\bar{y}(t) = y_0 + \int_{t_0}^t f(\tau, \bar{y}(\tau)) d\tau, \quad t \in [t_1, t_2].$$

Следовательно, функция $\bar{y}(t)$ удовлетворяет интегральному уравнению (2.3) при $t \in [t_1, t_2]$.

Пусть функция $\bar{y}(t)$ такова, что $\bar{y}(t) \in C[t_1, t_2]$, $|\bar{y}(t) - y_0| \leq A$ для $t \in [t_1, t_2]$ и $\bar{y}(t)$ удовлетворяет уравнению (2.3) для $t \in [t_1, t_2]$, то есть

$$\bar{y}(t) = y_0 + \int_{t_0}^t f(\tau, \bar{y}(\tau)) d\tau, \quad t \in [t_1, t_2]. \quad (2.4)$$

Покажем, что $\bar{y}(t)$ является решением задачи с начальным условием (2.1), (2.2).

Положив в (2.4) $t = t_0$, получим, что $\bar{y}(0) = y_0$. Следовательно условие (2.2) выполнено. Так как функция $\bar{y}(t)$ непрерывна на $[t_1, t_2]$, то правая часть равенства

$$\bar{y}(t) = y_0 + \int_{t_0}^t f(\tau, \bar{y}(\tau)) d\tau$$

непрерывно дифференцируема на $[t_1, t_2]$ как интеграл с переменным верхним пределом t от непрерывной функции $f(\tau, \bar{y}(\tau)) \in C[t_1, t_2]$. Следовательно, $\bar{y}(t)$ непрерывно дифференцируема на $[t_1, t_2]$. Дифференцируя (2.4), получим, что $\bar{y}(t)$ удовлетворяет (2.1), и лемма 2.1.1 доказана. $\square$

Гронуолла-Беллмана.

Лемма 2.1.2. Пусть функция $z(t) \in C[a, b]$ и такова, что

$$0 \leq z(t) \leq c + d \left| \int_{t_0}^t z(\tau) d\tau \right|, \quad t \in [a, b], \quad (2.5)$$

где постоянная c неотрицательна, постоянная d положительна, а t_0 – произвольное фиксированное число на отрезке $[a, b]$. Тогда

$$z(t) \leq ce^{d|t-t_0|}, \quad t \in [a, b]. \quad (2.6)$$

Доказательство. Рассмотрим $t \geq t_0$. Введем функцию

$$p(t) = \int_{t_0}^t z(\tau) d\tau, \quad t \in [t_0, b].$$

Тогда $p'(t) = z(t) \geq 0$, $p(t_0) = 0$. Из (2.5) следует, что $p'(t) \leq c + dp(t)$, $t \in [t_0, b]$. Умножив это неравенство на $e^{-d(t-t_0)}$, получим

$$p'(t)e^{-d(t-t_0)} \leq ce^{-d(t-t_0)} + dp(t)e^{-d(t-t_0)}, \quad t \in [t_0, b].$$

Это неравенство можно переписать так

$$\frac{d}{dt} \left(p(t)e^{-d(t-t_0)} \right) \leq ce^{-d(t-t_0)}, \quad t \in [t_0, b].$$

Проинтегрировав от t_0 до t , получим

$$p(t)e^{-d(t-t_0)} - p(t_0) \leq c \int_{t_0}^t e^{-d(\tau-t_0)} d\tau = \frac{c}{d} \left(1 - e^{-d(t-t_0)} \right), \quad t \in [t_0, b].$$

Учитывая то, что $p(t_0) = 0$, имеем $dp(t) \leq ce^{d(t-t_0)} - c$. Следовательно,

$$z(t) \leq c + dp(t) \leq c + ce^{d(t-t_0)} - c = ce^{d(t-t_0)}, \quad t \in [t_0, b]$$

и неравенство (2.6) для $t \in [t_0, b]$ доказано.

Докажем неравенство (2.6) для $t \in [a, t_0]$. Перепишем неравенство (2.5) следующим образом

$$0 \leq z(t) \leq c - d \int_{t_0}^t z(\tau) d\tau = c + d \int_t^{t_0} z(\tau) d\tau, \quad t \in [a, t_0].$$

Обозначим

$$q(t) = \int_t^{t_0} z(\tau) d\tau, \quad t \in [a, t_0].$$

Тогда $q'(t) = -z(t) \leq 0$, $q(t_0) = 0$. Из неравенства (2.5) следует, что $-q'(t) \leq c + dq(t)$, $t \in [a, t_0]$. Умножив это неравенство на $e^{-d(t_0-t)}$, получим

$$-q'(t)e^{-d(t_0-t)} \leq ce^{-d(t_0-t)} + dq(t)e^{-d(t_0-t)}, \quad t \in [a, t_0].$$

Это неравенство можно переписать так

$$-\frac{d}{dt} \left(q(t)e^{-d(t_0-t)} \right) \leq ce^{-d(t_0-t)}, \quad t \in [a, t_0].$$

Проинтегрировав от t до t_0 , получим

$$q(t)e^{-d(t_0-t)} - q(t_0) \leq c \int_t^{t_0} e^{-d(t_0-\tau)} d\tau = \frac{c}{d} (1 - e^{-d(t_0-t)}), \quad t \in [a, t_0].$$

Следовательно, $dq(t) \leq ce^{d(t_0-t)} - c$. А значит

$$z(t) \leq c + dq(t) \leq c + ce^{d(t_0-t)} - c = ce^{d|t-t_0|}, \quad t \in [a, t_0]$$

и неравенство (2.6) для $t \in [a, t_0]$ доказано, что и завершает доказательство леммы 2.1.2. $\square$

Определение 2.1.2. Функция $f(t, y)$, заданная в прямоугольнике Π , удовлетворяет в Π условию Липшица по y , если

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2|, \quad \forall (t, y_1), (t, y_2) \in \Pi,$$

где L – положительная постоянная.

Теорема 2.1.1. Пусть функция $f(t, y)$ непрерывна в Π и удовлетворяет в Π условию Липшица по y . Если $y_1(t)$, $y_2(t)$ – решения задачи Коши (2.1), (2.2) на отрезке $[t_1, t_2]$, то $y_1(t) = y_2(t)$ для $t \in [t_1, t_2]$.

Доказательство. Так как $y_1(t)$ и $y_2(t)$ – решения задачи Коши (2.1), (2.2), то из леммы 2.1.1 следует, что они являются решениями интегрального уравнения (2.3). То есть

$$\begin{aligned} y_1(t) &= y_0 + \int_{t_0}^t f(\tau, y_1(\tau)) d\tau, \quad t \in [t_1, t_2], \\ y_2(t) &= y_0 + \int_{t_0}^t f(\tau, y_2(\tau)) d\tau, \quad t \in [t_1, t_2]. \end{aligned}$$

Вычитая второе уравнение из первого и оценивая разность по модулю, получаем

$$\begin{aligned} |y_1(t) - y_2(t)| &= \left| \int_{t_0}^t f(\tau, y_1(\tau)) d\tau - \int_{t_0}^t f(\tau, y_2(\tau)) d\tau \right| \leq \\ &\leq \left| \int_{t_0}^t |f(\tau, y_1(\tau)) - f(\tau, y_2(\tau))| d\tau \right|, \quad t \in [t_1, t_2]. \end{aligned}$$

Используя условие Липшица, имеем

$$|y_1(t) - y_2(t)| \leq L \left| \int_{t_0}^t |y_1(\tau) - y_2(\tau)| d\tau \right|, \quad t \in [t_1, t_2].$$

Обозначив $z(t) = |y_1(t) - y_2(t)|$, перепишем последнее неравенство следующим образом

$$0 \leq z(t) \leq L \left| \int_{t_0}^t z(\tau) d\tau \right|, \quad t \in [t_1, t_2].$$

6. Локальная теорема существования решения задачи Коши

Перейдем к доказательству существования решения задачи с начальным условием. Следует отметить, что мы можем доказать теорему существования не на всем исходном отрезке $[t_0 - T, t_0 + T]$, а на некотором, вообще говоря, меньшем. Поэтому эта теорема часто называется локальной теоремой существования решения задачи Коши.

Теорема 2.1.2. Пусть функция $f(t, y)$ непрерывна в Π , удовлетворяет в Π условию Липшица по y и

$$|f(t, y)| \leq M, \quad (t, y) \in \Pi.$$

Тогда на отрезке $[t_0 - h, t_0 + h]$, где

$$h = \min\left\{T, \frac{A}{M}\right\},$$

существует функция $y(t)$ такая, что $y(t) \in C^1[t_0 - h, t_0 + h]$, $|y(t) - y_0| \leq A$, $t \in [t_0 - h, t_0 + h]$,

$$y'(t) = f(t, y(t)), \quad t \in [t_0 - h, t_0 + h], \quad (2.7)$$

$$y(t_0) = y_0. \quad (2.8)$$

Доказательство. Из леммы 2.1.1 следует, что для доказательства теоремы достаточно доказать существование функции $y(t) \in C[t_0 - h, t_0 + h]$ такой, что $|y(t) - y_0| \leq A$, $t \in [t_0 - h, t_0 + h]$, и являющейся решением интегрального уравнения

$$y(t) = y_0 + \int_{t_0}^t f(\tau, y(\tau)) d\tau, \quad t \in [t_0 - h, t_0 + h]. \quad (2.9)$$

Проведем доказательство, используя метод последовательных приближений. Рассмотрим последовательность функций $y_k(t)$, $k = 0, 1, 2, \dots$ таких, что $y_0(t) = y_0$,

$$y_{k+1}(t) = y_0 + \int_{t_0}^t f(\tau, y_k(\tau)) d\tau, \quad t \in [t_0 - h, t_0 + h], k = 0, 1, 2, \dots \quad (2.10)$$

Покажем, используя метод математической индукции, что для всех $k = 0, 1, 2, \dots$ выполнено

$$y_k(t) \in C[t_0 - h, t_0 + h], \quad |y_k(t) - y_0| \leq A, \quad t \in [t_0 - h, t_0 + h].$$

Для $k = 0$ это очевидно справедливо, поскольку $y_0(t) = y_0$.

Пусть это верно для $k = m$. То есть

$$y_m(t) \in C[t_0 - h, t_0 + h], \quad |y_m(t) - y_0| \leq A, \quad t \in [t_0 - h, t_0 + h].$$

Покажем, что

$$y_{m+1}(t) = y_0 + \int_{t_0}^t f(\tau, y_m(\tau)) d\tau, \quad t \in [t_0 - h, t_0 + h] \quad (2.11)$$

такова, что $y_{m+1}(t) \in C[t_0 - h, t_0 + h]$ и $|y_{m+1}(t) - y_0| \leq A$, $t \in [t_0 - h, t_0 + h]$.

Действительно, так как $|y_m(t) - y_0| \leq A$, $t \in [t_0 - h, t_0 + h]$, то функция $f(t, y_m(t))$ определена и непрерывна на $[t_0 - h, t_0 + h]$. Значит интеграл, стоящий в правой части (2.11), определен и непрерывен при $t \in [t_0 - h, t_0 + h]$. Следовательно, $y_{m+1}(t) \in C[t_0 - h, t_0 + h]$.

Оценим

$$\begin{aligned}
|y_{m+1}(t) - y_0| &= \left| \int_{t_0}^t f(\tau, y_m(\tau)) d\tau \right| \leq \\
&\leq \left| \int_{t_0}^t |f(\tau, y_m(\tau))| d\tau \right| \leq \left| \int_{t_0}^t M d\tau \right| \leq Mh \leq M \cdot \frac{A}{M} = A, \quad t \in [t_0 - h, t_0 + h].
\end{aligned}$$

Таким образом $|y_{m+1}(t) - y_0| \leq A$, $t \in [t_0 - h, t_0 + h]$. Следовательно, мы показали что все $y_k(t) \in C[t_0 - h, t_0 + h]$ и $|y_k(t) - y_0| \leq A$, $t \in [t_0 - h, t_0 + h]$, $k = 0, 1, 2, \dots$.

Докажем, используя метод математической индукции, что для $t \in [t_0 - h, t_0 + h]$ справедливы неравенства

$$|y_{k+1}(t) - y_k(t)| \leq AL^k \frac{|t - t_0|^k}{k!}, \quad k = 0, 1, 2, \dots \quad (2.12)$$

Для $k = 0$ имеем

$$\begin{aligned}
|y_1(t) - y_0(t)| &= \left| y_0 + \int_{t_0}^t f(\tau, y_0) d\tau - y_0 \right| \leq \\
&\leq \left| \int_{t_0}^t f(\tau, y_0) d\tau \right| \leq Mh \leq A, \quad t \in [t_0 - h, t_0 + h],
\end{aligned}$$

то есть при $k = 0$ оценка (2.12) верна.

Пусть неравенство (2.12) справедливо для $k = m - 1$. Покажем, что тогда оно справедливо при $k = m$. Действительно

$$\begin{aligned}
|y_{m+1}(t) - y_m(t)| &= \left| y_0 + \int_{t_0}^t f(\tau, y_m(\tau)) d\tau - y_0 - \int_{t_0}^t f(\tau, y_{m-1}(\tau)) d\tau \right| \leq \\
&\leq \left| \int_{t_0}^t |f(\tau, y_m(\tau)) - f(\tau, y_{m-1}(\tau))| d\tau \right|, \quad t \in [t_0 - h, t_0 + h].
\end{aligned}$$

Используя условие Липшица и неравенство (2.12) для $k = m - 1$, получим

$$\begin{aligned} |y_{m+1}(t) - y_m(t)| &\leq L \left| \int_{t_0}^t |y_m(\tau) - y_{m-1}(\tau)| d\tau \right| \leq \\ &\leq L \left| \int_{t_0}^t AL^{m-1} \frac{|\tau - t_0|^{m-1}}{(m-1)!} d\tau \right| = AL^m \frac{|t - t_0|^m}{m!}, \quad t \in [t_0 - h, t_0 + h]. \end{aligned}$$

Следовательно, оценка (2.12) справедлива при $k = m$, и значит она доказана для любого $k \in \mathbb{N}$.

Представим функции $y_k(t)$ как частичные суммы ряда

$$y_k(t) = y_0 + \sum_{n=1}^k (y_n(t) - y_{n-1}(t)), \quad n = 1, 2, \dots$$

Равномерная сходимость последовательности функций $y_k(t)$ на отрезке $[t_0 - h, t_0 + h]$ эквивалентна равномерной сходимости ряда

$$\sum_{n=1}^{\infty} (y_n(t) - y_{n-1}(t)) \quad (2.13)$$

на отрезке $[t_0 - h, t_0 + h]$. Применим признак Вейерштрасса для доказательства равномерной сходимости ряда (2.13) на отрезке $[t_0 - h, t_0 + h]$. Из оценки (2.12) следует, что

$$|y_n(t) - y_{n-1}(t)| \leq AL^{n-1} \frac{h^{n-1}}{(n-1)!} = c_n, \quad t \in [t_0 - h, t_0 + h], \quad n = 1, 2, \dots$$

Числовой ряд $\sum_{n=1}^{\infty} c_n$ сходится по признаку Даламбера. Следовательно, ряд (2.13) сходится равномерно на отрезке $[t_0 - h, t_0 + h]$. Это означает, что последовательность функций $y_k(t)$ сходится равномерно на отрезке $[t_0 - h, t_0 + h]$ к некоторой функции $y(t)$. Так как все функции $y_k(t)$ непрерывны на отрезке $[t_0 - h, t_0 + h]$, то функция $y(t)$ также непрерывна на этом отрезке, то есть $y(t) \in C[t_0 - h, t_0 + h]$.

Покажем, что $|y(t) - y_0| \leq A$, $t \in [t_0 - h, t_0 + h]$. Как было доказано, $|y_k(t) - y_0| \leq A$, $t \in [t_0 - h, t_0 + h]$, $k = 0, 1, 2, \dots$. Переходя в этом

неравенстве к пределу при $k \rightarrow \infty$ и произвольном фиксированном $t \in [t_0 - h, t_0 + h]$, получим, что $|y(t) - y_0| \leq A$, $t \in [t_0 - h, t_0 + h]$.

Покажем, что $y(t)$ является решением интегрального уравнения (2.9). В силу равномерной на отрезке $[t_0 - h, t_0 + h]$ сходимости $y_k(t)$ к функции $y(t)$ для произвольного $\delta > 0$ найдется номер $k_0(\delta)$ такой, что при $k \geq k_0(\delta)$ справедливо неравенство $|y_k(t) - y(t)| < \delta$ для всех $t \in [t_0 - h, t_0 + h]$. Тогда для любого $\varepsilon > 0$ выбираем $\delta(\varepsilon) = \frac{\varepsilon}{Lh}$ и $k_0 = k_0(\delta(\varepsilon))$ так, что при $k \geq k_0$ справедливо неравенство

$$|f(\tau, y_k(\tau)) - f(\tau, y(\tau))| \leq L|y_k(\tau) - y(\tau)| < \frac{\varepsilon}{h}, \quad \tau \in [t_0 - h, t_0 + h].$$

Тогда для разности интегралов получаем оценки

$$\left| \int_{t_0}^t f(\tau, y_k(\tau)) d\tau - \int_{t_0}^t f(\tau, y(\tau)) d\tau \right| < \frac{\varepsilon}{h} |t - t_0| \leq \varepsilon, \quad t \in [t_0 - h, t_0 + h],$$

позволяющие перейти в (2.10) к пределу при $k \rightarrow \infty$ и произвольном фиксированном $t \in [t_0 - h, t_0 + h]$. В результате получаем, что $y(t)$ является решением интегрального уравнения (2.9).

Таким образом, мы показали, что $y(t) \in C[t_0 - h, t_0 + h]$, $|y(t) - y_0| \leq A$, $t \in [t_0 - h, t_0 + h]$ и является решением интегрального уравнения (2.9). Следовательно, $y(t)$ является решением задачи с начальным условием на отрезке $[t_0 - h, t_0 + h]$ и теорема 2.1.2 доказана. $\square$

24. *Функции алгебры логики. Реализация их формулами. Совершенная дизъюнктивная нормальная форма.*

1. Пусть $U = \{u_1, u_2, \dots, u_n, \dots\}$ — исходный алфавит переменных (аргументов). Через E_2 обозначим множество $\{0, 1\}$.

Определение. Функция $f(u_{i_1}, u_{i_2}, \dots, u_{i_n})$ называется функцией алгебры логики (или булевой функцией) тогда и только тогда, когда все ее аргументы определены на множестве E_2 и все свои значения функция принимает из множества E_2 .

Иными словами, если $(\alpha_1, \alpha_2, \dots, \alpha_n)$ есть некоторый набор значений переменных $(u_{i_1}, u_{i_2}, \dots, u_{i_n})$ такой, что для любого $i \in \{1, 2, \dots, n\}$ $\alpha_i \in E_2$, то $f(\alpha_1, \alpha_2, \dots, \alpha_n) \in E_2$.

В дальнейшем для упрощения обозначений вместо символов переменных из алфавита U мы будем использовать строчные латинские буквы $(x, y, z, \dots)$ с индексами или без них.

Множество всех функций алгебры логики обозначим через P_2 .

Для задания функции алгебры логики $f(x_1, x_2, \dots, x_n)$ достаточно указать значение функции на каждом из 2^n наборов значений переменных $(x_1, x_2, \dots, x_n)$. Поставим в соответствие каждому набору $(\alpha_1, \alpha_2, \dots, \alpha_n)$ целое неотрицательное число $v(\alpha_1, \alpha_2, \dots, \alpha_n) = \sum_{i=1}^n \alpha_i \cdot 2^{n-i}$

(набор $(\alpha_1, \alpha_2, \dots, \alpha_n)$ представляет собой двоичную запись числа $v(\alpha_1, \alpha_2, \dots, \alpha_n)$) и упорядочим все наборы по возрастанию соответствующих им чисел $v(\alpha_1, \alpha_2, \dots, \alpha_n)$ от 0 до $2^n - 1$. Такой порядок следования наборов называется лексикографическим. Теперь функцию $f(x_1, x_2, \dots, x_n)$ можно задать с помощью таблицы 1 (наборы значений переменных следуют в лексикографическом порядке). Различные таблицы, определяющие функции алгебры логики, зависящие от фиксированных n переменных, будут различаться лишь своим последним столбцом. Справедлива следующая

Теорема. Число функций алгебры логики, зависящих от n фиксированных переменных, равно 2^{2^n} .

Доказательство следует из того, что значение функции на каждом из 2^n наборов можно определить одним из двух способов (0 или 1) независимо от значений функции на других наборах.

Таблица 1.

x_1	x_2	...	x_{n-1}	x_n	$f(x_1, x_2, \dots, x_{n-1}, x_n)$
0	0	...	0	0	$f(0, 0, \dots, 0, 0)$
0	0	...	0	1	$f(0, 0, \dots, 0, 1)$
...	...	...	...	...	...
α_1	α_2	...	α_{n-1}	α_n	$f(\alpha_1, \alpha_2, \dots, \alpha_{n-1}, \alpha_n)$
...	...	...	...	...	...
1	1	...	1	1	$f(1, 1, \dots, 1, 1)$

Определение. Функция $f(x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n)$ существенно зависит от переменной x_i тогда и только тогда, когда существуют такие числа $\alpha_1, \alpha_2, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_{n-1}, \alpha_n$ из E_2 , что $f(\alpha_1, \alpha_2, \dots, \alpha_{i-1}, 0, \alpha_{i+1}, \dots, \alpha_{n-1}, \alpha_n) \neq f(\alpha_1, \alpha_2, \dots, \alpha_{i-1}, 1, \alpha_{i+1}, \dots, \alpha_{n-1}, \alpha_n)$. При этом переменная x_i называется существенной переменной функции $f(x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n)$.

Переменная, не являющаяся существенной, называется несущественной или фиктивной.

Пусть переменная x_i является фиктивной переменной функции $f(x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n)$. По таблице, задающей функцию f , построим новую таблицу путем вычеркивания всех строк вида $(\alpha_1, \alpha_2, \dots, \alpha_{i-1}, 1, \alpha_{i+1}, \dots, \alpha_{n-1}, \alpha_n)$ и столбца переменной x_i . Новая таблица задает некоторую функцию $g(x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$. Будем говорить, что функция $g(x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$ получена из функции $f(x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n)$ путем удаления фиктивной переменной x_i , а функция $f(x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n)$ получена из функции $g(x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$ путем введения фиктивной переменной x_i .

Таблица 2.

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	1	0	1
1	0	1	1	0

Таблица 3.

x	y	$f_5(x,y)$	$f_6(x,y)$	$f_7(x,y)$	$f_8(x,y)$	$f_9(x,y)$	$f_{10}(x,y)$
0	0	0	0	1	0	1	1
0	1	0	1	1	1	0	1
1	0	0	1	0	1	0	1
1	1	1	1	1	0	1	0

Определение. Функции f и g называются равными, если одна из них может быть получена из другой путем удаления и введения фиктивных переменных.

Рассмотрим в качестве примера так называемые «элементарные» функции алгебры логики (см. таблицы 2 и 3).

Функции $f_1(x)$ и $f_2(x)$ — это константы 0 и 1 соответственно. Заметим, что у них нет существенных переменных. Функция $f_4(x) = (x)$ называется тождественной. Функция $f_4(x) = \bar{x}$ называется отрицанием x , запись $\bar{x}$ читается «не x ». Функция $f_5(x,y) = (x \& y)$ называется конъюнкцией x и y , запись $x \& y$ читается « x и y ». Иногда в качестве знака конъюнкции используют точку умножения, иногда знак вообще опускается, и пишут xy . Функция $f_6(x,y) = (x \vee y)$ называется дизъюнкцией x и y , запись $x \vee y$ читается « x или y ». Функция $f_7(x,y) = (x \rightarrow y)$ называется импликацией x и y , запись $x \rightarrow y$ читается «из x следует y ». Функция $f_8(x,y) = (x \oplus y)$ называется сложением x и y по модулю 2, запись $x \oplus y$ читается « x плюс y ». Функция $f_9(x,y) = (x \sim y)$ называется эквивалентностью x и y , запись $x \sim y$ читается « x эквивалентен y ». Функция $f_{10}(x,y) = (x|y)$ называется штрихом Шеффера, запись $x|y$ читается «штрих Шеффера x и y ».

2. Кроме описанного выше табличного способа задания булевых функций существует и способ задания посредством формул. Дадим индуктивное определение формул.

Определение.

Пусть P — некоторое множество функций алгебры логики.

Базис индукции. Каждая функция $f(x_1, x_2, \dots, x_n)$ из P называется формулой над P .

Шаг индукции. Пусть $f(x_1, x_2, \dots, x_n)$ — функция из P и $A_1, A_2, \dots, A_n$ — либо формулы над P , либо символы переменных. Тогда выражение $f(A_1, A_2, \dots, A_n)$ называется формулой над P .

Формулы, используемые для построения формулы C , называются подформулами формулы C .

Проводя индукцию аналогично тому, как это было сделано в определении формулы, поставим каждой формуле в соответствие функцию.

Определение.

Базис индукции. Пусть формула C имеет вид $f(x_1, x_2, \dots, x_n)$, где $f \in P$. Тогда формуле C поставим в соответствие функцию $f(x_1, x_2, \dots, x_n)$.

Шаг индукции. Пусть $f(x_1, x_2, \dots, x_n)$ — функция из P и A_i ($i=1, 2, \dots, n$) — либо формула над P , которой по предположению индукции поставлена в соответствие функция f_i , либо символ переменной $x_{f(i)}$, которому соответствует функция $f_i = x_{f(i)}$. Тогда формуле C , имеющей вид $f(A_1, A_2, \dots, A_n)$, поставим в соответствие функцию $f(f_1, f_2, \dots, f_n)$.

Если формуле C соответствует функция f , то говорят, что формула C реализует функцию f . Функция f , соответствующая формуле над P , называется суперпозицией функций из P , а процесс получения f из P — операцией суперпозиции.

Формулы называются эквивалентными тогда и только тогда, когда соответствующие им функции равны.

Приведем список некоторых эквивалентностей над системой элементарных функций алгебры логики. Пусть запись $(x \circ y)$ обозначает какую-либо из функций $(x \& y)$, $(x \vee y)$, $(x \oplus y)$, $(x \sim y)$, $(x | y)$ (фиксированную для одной эквивалентности).

1) Функция $(x \circ y)$ обладает свойством коммутативности:

$$(x \circ y) = (y \circ x).$$

2) Функция $(x \circ y)$ обладает свойством ассоциативности:

$$((x \circ y) \circ z) = (x \circ (y \circ z)).$$

3) Имеют место свойства дистрибутивности для конъюнкции и дизъюнкции:

$$((x \vee y) \& z) = ((x \& z) \vee (y \& z)), \quad ((x \& y) \vee z) = ((x \vee z) \& (y \vee z)).$$

4) Имеют место следующие соотношения:

$$\overline{\overline{x}} = x \text{ (правило снятия двойного отрицания),}$$

$$\overline{(x \& y)} = (\overline{x} \vee \overline{y}), \quad \overline{(x \vee y)} = (\overline{x} \& \overline{y}) \text{ (правила де Моргана),}$$

$$((x \vee y) \& x) = x, \quad ((x \& y) \vee x) = x \text{ (правила поглощения),}$$

$$(x \& x) = x, \quad (x \vee x) = x, \quad (x \& 1) = x, \quad (x \& 0) = 0, \quad (x \vee 0) = x, \quad (x \vee 1) = 1, \quad (x \& \overline{x}) = 0,$$

$$(x \vee \overline{x}) = 1.$$

В справедливости всех перечисленных эквивалентностей можно убедиться непосредственно.

В дальнейшем при записи формул над множеством элементарных функций будем использовать нижеследующие соглашения:

1) в подформулах, представляющих собой формулы над функцией $(x_1 \circ x_2)$, обладающей свойством ассоциативности, внутренние скобки опускаются (например, вместо $(x_1 \circ (x_1 \circ x_2))$ пишется $(x_1 \circ x_1 \circ x_2)$),

2) внешние скобки в записи формулы опускаются,

3) конъюнкция считается сильнее других двуместных операций, и в подформулах, представляющих собой формулы над функцией $x_1 \& x_2$, внешние скобки опускаются (например, вместо $(x_1 \vee (x_1 \& x_2))$ пишется $(x_1 \vee x_1 \& x_2)$),

4) внешние скобки в записи подформулы, над которой стоит символ отрицания, опускаются (например, вместо $\overline{(x \rightarrow y)}$ пишется $\overline{x \rightarrow y}$).

Выражения, получающиеся из формул при применении указанных соглашений, вообще говоря, не являются формулами, но мы будем употреблять термин «формула» и по отношению к этим выражениям.

Введем в употребление следующее обозначение:

$$\bigvee_{i=1}^t x_i = x_1 \vee x_2 \vee \dots \vee x_t.$$

3. Изучим некоторые специальные представления функции алгебры логики.

Введем следующее обозначение

$$x^\sigma = \begin{cases} \bar{x} & \text{при } \sigma = 0, \\ x & \text{при } \sigma = 1, \end{cases}$$

где σ — параметр. Заметим, что $\sigma^\sigma = 1$, а $\sigma^{\bar{\sigma}} = 0$.

Теорема о разложении функции по переменным. Каждую функцию алгебры логики $f(x_1, x_2, \dots, x_m, x_{m+1}, \dots, x_n)$ при любом m ($1 \leq m \leq n$) можно представить в виде

$$\begin{aligned} & f(x_1, x_2, \dots, x_m, x_{m+1}, \dots, x_n) = \\ & = \bigvee_{(\sigma_1, \dots, \sigma_m)} x_1^{\sigma_1} \& \dots \& x_m^{\sigma_m} \& f(\sigma_1, \sigma_2, \dots, \sigma_m, x_{m+1}, \dots, x_n), \end{aligned} \quad (*)$$

где дизъюнкция берется по всевозможным наборам значений переменных $x_1, x_2, \dots, x_m$.

Доказательство.

Рассмотрим произвольный набор $(\alpha_1, \alpha_2, \dots, \alpha_m, \alpha_{m+1}, \dots, \alpha_n)$ значений переменных. В левой части соотношения (*) получим $f(\alpha_1, \alpha_2, \dots, \alpha_m, \alpha_{m+1}, \dots, \alpha_n)$. Правая часть преобразуется к виду

$$\bigvee_{(\sigma_1, \dots, \sigma_m)} \alpha_1^{\sigma_1} \& \dots \& \alpha_m^{\sigma_m} \& f(\sigma_1, \sigma_2, \dots, \sigma_m, \alpha_{m+1}, \dots, \alpha_n).$$

Пусть числа $\sigma_1, \sigma_2, \dots, \sigma_m$ таковы, что хотя бы для одного $i \in \{1, \dots, m\}$

$\alpha_i \neq \sigma_i$. Тогда $\alpha_i^{\sigma_i} = 0$, и соответствующая набору $(\sigma_1, \sigma_2, \dots, \sigma_m)$ конъюнкция обращается в нуль. Пусть теперь числа $\sigma_1, \sigma_2, \dots, \sigma_m$ та-

ковы, что для каждого $i \in \{1, \dots, m\}$ $\alpha_i = \sigma_i$. Тогда все $\alpha_i^{\sigma_i} = 1$, и соответствующая набору $(\sigma_1, \sigma_2, \dots, \sigma_m)$ конъюнкция

равна $f(\sigma_1, \sigma_2, \dots, \sigma_m, \alpha_{m+1}, \dots, \alpha_n) = f(\alpha_1, \alpha_2, \dots, \alpha_m, \alpha_{m+1}, \dots, \alpha_n)$. Значит, правая часть соотношения

(*) принимает то же значение, что и левая. Теорема доказана. $\square$

Выделим два крайних случая ($m=1$ и $m=n$) и сформулируем следствия данной теоремы.

Следствие 1 (разложение функции по одной переменной).

$$f(x_1, x_2, \dots, x_n) = \overline{x_1} \& f(0, x_2, \dots, x_n) \vee x_1 \& f(1, x_2, \dots, x_n).$$

Следствие 2 (разложение функции в совершенную дизъюнктивную нормальную форму). Любая функция алгебры логики $f(x_1, x_2, \dots, x_n)$, тождественно не равная нулю¹, может быть представлена в следующем виде:

$$f(x_1, x_2, \dots, x_n) = \bigvee_{\substack{(\sigma_1, \dots, \sigma_n), \\ f(\sigma_1, \dots, \sigma_n)=1}} x_1^{\sigma_1} \& \dots \& x_n^{\sigma_n}.$$

(правая часть последнего соотношения называется совершенной дизъюнктивной нормальной формой).

Определение. Система функций P называется полной в P_2 тогда и только тогда, когда любая функция алгебры логики может быть реализована формулой над P .

Теорема. Система функций $\{\overline{x}, x\&y, x\vee y\}$ является полной в P_2 .

Доказательство.

Любая тождественно не равная нулю функция алгебры логики может быть реализована на основании следствия 2 формулой над $\{\overline{x}, x\&y, x\vee y\}$, представляющей совершенную дизъюнктивную нормальную форму. Если же функция тождественно равна нулю, то ее можно реализовать формулой $x\&\overline{x}$. Теорема доказана. $\square$

25. Схемы из функциональных элементов и простейшие алгоритмы их синтеза. Оценка сложности схем, получаемых по методу Шеннона.

Определение. *Графом G называется любая пара (V, E) , где $V = \{v_1, v_2, \dots\}$ — множество элементов любой природы, а $E = \{e_1, e_2, \dots\}$ — семейство пар элементов из V , причем допускаются пары вида (v_i, v_i) и одинаковые пары. Если пары в V рассматриваются как пары неупорядоченных элементов, то граф называется *неориентированным*, если же как пары упорядоченных элементов, то граф называется *ориентированным (орграфом)*.*

Элементы множества V называются *вершинами* графа, а пары из E — его *ребрами*, в орграфе они называются *ориентированными ребрами* или *дугами*. Пара вида (v_i, v_i) называется *петлей* в вершине v_i . Если пара (v_i, v_j) встречается в E более одного раза, то говорят, что (v_i, v_j) — *кратное ребро*. Говорят, что ребро $e = (u, v)$ в неориентированном графе *соединяет* вершины u и v , а в ориентированном графе дуга $e = (u, v)$ *идет* из вершины u в вершину v .

Графы можно условно изображать следующим образом. Вершины изображаются различными точками, а каждое ребро (дуга) (v_i, v_j) из E — линией, соединяющей точки, соответствующие v_i и v_j . Если (v_i, v_j) — дуга, то на этой линии ставится стрелка от v_i к v_j .

Определение. Говорят, что вершины v_i и v_j *смежны* в графе $G = (V, E)$, если в E входит пара (v_i, v_j) или (v_j, v_i) . Говорят, что ребро (дуга) (v_i, v_j) *инцидентно* вершинам v_i, v_j .

Определение. *Степенью вершины* в графе называется число ребер, инцидентных данной вершине, при этом петли учитываются дважды. Вершины степени 0 называют *изолированными*. *Полустепенью исхода $d^-(v)$ (полустепенью захода $d^+(v)$)* вершины v в ориентированном графе называется число дуг, выходящих из данной вершины (соответственно, входящих в данную вершину).

Определение. Пусть $G = (V, E)$ и $G' = (V', E')$ — два графа. Тогда G и G' называются *изоморфными*, если существуют взаимно однозначные отображения $\varphi: V \rightarrow V'$, $\psi: E \rightarrow E'$ такие, что для любого ребра (дуги) (u, v) из E выполняется равенство: $\psi(u, v) = (\varphi(u), \varphi(v))$. Иными словами, отображения φ и ψ должны сохранять отношение инцидентности вершин и ребер (дуг).

Определение. Граф $G' = (V', E')$ называется *подграфом* графа $G = (V, E)$, если $V' \subseteq V$ и $E' \subseteq E$.

Определение. *Путь* в графе (орграфе) $G = (V, E)$ называется последовательность P вершин и ребер (дуг) вида $v_0, (v_0, v_1), v_1, \dots, v_{n-1}, (v_{n-1}, v_n), v_n$, где каждая вершина v_i лежит в V и каждое ребро (дуга) (v_i, v_{i+1}) лежит в E . *Длина пути* — это число ребер (дуг) в нем. Говорят, что путь P идет из v_0 в v_n . *Цепь* — это путь без повторяющихся ребер (дуг), *простая цепь* — путь без повторяющихся вершин. Путь называется *замкнутым*, если $v_n = v_0$. Замкнутый путь называется *циклом*, если ребра (дуги) в нем не повторяются. Замкнутый путь называется *простым циклом*, если, за исключением совпадения v_n и v_0 , других повторений вершин в нем нет.

Замечание. В случае орграфов во избежание неясностей вместо слов «путь», «цикл» также будут использоваться выражения «ориентированный путь», «ориентированный цикл», соответственно.

Далее будут рассматриваться только конечные графы, то есть графы с конечным числом вершин и ребер (дуг).

Определение. *Ориентированным ациклическим графом* называется любой ориентированный граф, в котором нет ориентированных циклов.

Определение. Вершина ориентированного графа называется *истоком* (*стоком*), если в нее не входит ни одна дуга, то есть $d^+(v) = 0$, (соответственно, из нее не выходит ни одной дуги, то есть $d^-(v) = 0$).

Лемма 1. В любом (конечном) ориентированном ациклическом графе есть хотя бы один исток и хотя бы один сток.

Доказательство. Выберем любую вершину и будем строить путь из нее, двигаясь произвольно по направлению дуг. При этом вершины не могут повторяться, так как в данном орграфе нет ориентированных циклов. Поэтому в силу конечности графа путь должен прийти в тупик, который и является стоком. Для получения истока надо аналогично строить путь, двигаясь против направления дуг.

Лемма 1 позволяет ввести следующее

Определение. *Глубиной вершины v* в ориентированном ациклическом графе называется максимум длин ориентированных путей, начинающихся в истоках графа и заканчивающихся в вершине v . Очевидно, что глубина каждой вершины в ориентированном ациклическом графе не превосходит $p - 1$, где p — число вершин в графе.

Лемма 2. Пусть (u, v) — дуга в ориентированном ациклическом графе. Тогда глубина вершины v больше, чем глубина вершины u .

Доказательство сразу следует из того, что если существует ориентированный путь длины k из некоторого истока в вершину u , то из того же истока в вершину v существует ориентированный путь длины $k+1$.

Пусть X и Z — счетные упорядоченные попарно не пересекающиеся алфавиты булевых переменных, причем булевы переменные из X (из Z) считаются *входными* (соответственно, *выходными*) булевыми переменными. Пусть $B, B = \{\varphi_i \mid i = 1, \dots, b\}$, — полное в P_2 множество функций алгебры логики, где функция алгебры логики (ФАЛ) $\varphi_i, i = 1, \dots, b$, зависит от $k_i, k_i \geq 1$, переменных и при $k_i \geq 2$ существенно зависит от всех своих переменных.

Ориентированный граф G называется *упорядоченным*, если дуги, входящие в каждую его вершину v , упорядочены, т. е. пронумерованы числами $1, 2, \dots, d^+(v)$.

Определение. Схемой из функциональных элементов (СФЭ) над базисом B называется ориентированный ациклический упорядоченный граф Σ , вершины которого помечены следующим образом:

1) каждый исток Σ помечен символом некоторой переменной из X , причем различные истоки помечены символами различных переменных;

2) каждая отличная от истока вершина v схемы Σ помечена символом некоторой функции φ_i из множества B и при этом $k_i = d^+(v)$ (такая вершина с приписанным символом φ_i представляет функциональный элемент (ФЭ) E_i с k_i входами и одним выходом, для наглядности такие ФЭ будем изображать в виде треугольников с записанным внутри них символом φ_i , как на рис. 1);

3) некоторые вершины Σ помечены символами переменных из Z так, что одной и той же вершине может быть сопоставлено несколько переменных из Z , но разным вершинам не может быть сопоставлена одна и та же переменная. При этом входные (выходные) переменные, которые приписаны каким-либо вершинам Σ , считаются входными (соответственно, выходными) переменными Σ , а те вершины, которым они сопоставлены, — входами (соответственно, выходами) СФЭ Σ .

Определим теперь функционирование СФЭ Σ над базисом B входными переменными $x_1, x_2, \dots, x_n$ и выходными переменными $z_1, z_2, \dots, z_m$. Индукцией по $q, q = 0, 1, \dots$, найдем для каждой вершины v глубины q в схеме Σ реализуемую в ней функцию $F_v = F_v(x_1, x_2, \dots, x_n)$. Если $q = 0$, т. е. v — вход Σ , положим $F_v = x_j$, где x_j — входная переменная, сопоставленная вершине v . Пусть теперь v — вершина глубины $q, q \geq 1$, схемы Σ , которая имеет пометку ϕ_i и в которую входит k_i дуг, причем дуга с номером $j, 1 \leq j \leq k_i$, исходит из вершины v_j глубины q_j , где уже реализована функция F_{v_j} . Тогда в вершине v реализуется функция алгебры логики $F_v = F_v(x_1, x_2, \dots, x_n)$, задаваемая суперпозицией $F_v = \phi_i(F_{v_1}, F_{v_2}, \dots, F_{v_{k_i}})$. Пусть символы выходных переменных $z_1, z_2, \dots, z_m$ приписаны не обязательно различным вершинам $u_1, u_2, \dots, u_m$. Тогда будем считать, что СФЭ Σ реализует систему функций $F = \{F_{u_1}, F_{u_2}, \dots, F_{u_m}\}$.

Схемы, реализующие одинаковые системы ФАЛ, называются эквивалентными.

Две СФЭ Σ' и Σ'' называются изоморфными, если они изоморфны как помеченные упорядоченные графы, т. е. если существуют взаимно однозначные отображения множества вершин Σ' на множество вершин Σ'' и множества дуг Σ' на множество дуг Σ'' , сохраняющие отношение инцидентности вершин и дуг, а также сохраняющие все номера дуг, заходящих в вершины, и все пометки. Из определения вытекает, что в соответствующих друг другу вершинах изоморфных СФЭ реализуются одинаковые функции. Следовательно, изоморфные СФЭ эквивалентны.

Замечание. Изменение нумерации дуг, входящих в такую вершину v СФЭ Σ , которой сопоставлен ФЭ E , с симметрической (то есть не меняющейся при произвольных перестановках переменных) ФАЛ ϕ_i , не изменяет ФАЛ, реализуемую в вершине v , а значит, не влияет на функционирование Σ . В связи с этим в подобных случаях номера дуг, входящих в вершину v , могут не указываться.

Вершина СФЭ называется *висячей*, если из нее не выходит ни одна дуга и она не является выходом схемы. При удалении из СФЭ *вершины* удаляются и все инцидентные этой вершине дуги. Две вершины СФЭ считаются *эквивалентными*, если в них реализуются равные ФАЛ.

Пусть СФЭ Σ' получается из СФЭ Σ в результате выполнения следующих действий: 1) переноса начальной вершины всех дуг Σ , исходящих из вершины v , а также всех выходных переменных, приписанных v , в отличную от нее эквивалентную вершину w , глубина которой в Σ не больше, чем глубина v , 2)

удаления вершины v . Тогда СФЭ Σ' считается результатом применения к СФЭ Σ *операции исключения вершины v эквивалентным преемником w* . Ясно, что схемы Σ и Σ' эквивалентны.

Схема называется *приведенной (строго приведенной)*, если в ней нет висячих (соответственно висячих и эквивалентных) вершин. Заметим, что из любой СФЭ можно получить эквивалентную ей приведенную (строго приведенную) СФЭ с помощью операции удаления висячих (соответственно, с помощью операций удаления висячих и исключения эквивалентных) вершин.

Число функциональных элементов в СФЭ Σ называется ее *сложностью* и обозначается через $L(\Sigma)$. Для системы ФАЛ F через $L_B(F)$ обозначим минимум сложности таких СФЭ Σ над базисом B , которые реализуют F . Величина $L_B(F)$ называется *сложностью системы ФАЛ F в классе СФЭ над базисом B* . При этом СФЭ Σ над базисом B , которая реализует F и для которой $L(\Sigma)=L_B(F)$, называется *минимальной*. Введем функцию $L_B(n)$ натурального аргумента n , называемую *функцией Шеннона для класса СФЭ над базисом B* и определяемую следующим образом:

$$L_B(n) = \max_{f(x_1, \dots, x_n) \in P_2^n} L_B(f(x_1, \dots, x_n))$$

(через P_2^n здесь обозначено множество всех ФАЛ от переменных $x_1, x_2, \dots, x_n$). Функция Шеннона характеризует максимум сложности минимальной СФЭ над базисом B по всем ФАЛ от n фиксированных переменных.

В дальнейшем будут рассматриваться только СФЭ в базисе $B_0 = \{\varphi_1 = x_1 \& x_2, \varphi_2 = x_1 \vee x_2, \varphi_3 = \bar{x}_1\}$ (см. рис. 1), поэтому индекс $B = B_0$ будет опускаться, а о СФЭ иногда будет упоминаться без указания базиса.

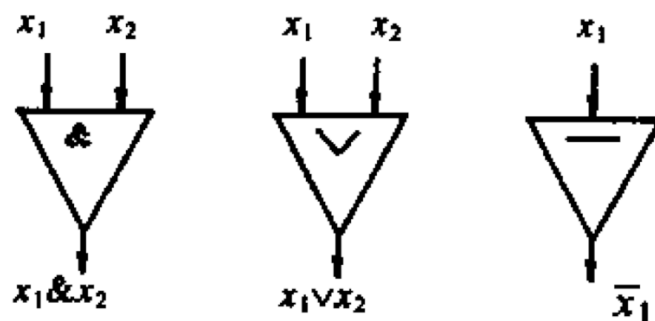


Рис. 1. Функциональные элементы базиса B_0

Для решения задачи синтеза СФЭ можно использовать переборный алгоритм, который просматривает все схемы сложности 1, 2, 3, ... до тех пор, пока не найдет схему, реализующую заданную систему ФАЛ. Следует отметить, что трудоемкость этого метода синтеза очень быстро растет с ростом числа переменных n , и что при $n \geq 5$ он становится практически неприменимым.

Для тождественно не равных нулю ФАЛ можно использовать элементарный метод синтеза, основанный на моделировании совершенной ДНФ. С учетом очевидного тождества $0 = x \& \bar{x}$ получаем, что имеет место

Лемма 3. Для любой ФАЛ $f(x_1, x_2, \dots, x_n)$ существует СФЭ в базисе B_0 , реализующая f со сложностью не более, чем $n \cdot 2^{n+1}$.

Для множества A и любого натурального t через A^t обозначается t -я декартова степень множества A , т. е. множество всех наборов длины t с элементами из A . Для набора β , $\beta \in A^t$, через $\beta[i]$, где $0 \leq i \leq t-1$, обозначается его i -й (начиная с нулевого) элемент, т.

е. $\beta = (\beta[0], \dots, \beta[t-1])$. Число $\sum_{i=0}^{t-1} \beta[i] \cdot 2^{t-i-1}$, двоичная запись

которого совпадает с β , называется номером β и обозначается через $|\beta|$.

Система ФАЛ Q_n из P_2^n такая, что при всех $i, i = 0, 1, 2, \dots, 2^n - 1$, справедливо равенство:

$$Q_n[i] = x_1^{\sigma_1} \cdot x_2^{\sigma_2} \cdot \dots \cdot x_n^{\sigma_n}, \quad (1)$$

где $\sigma = (\sigma_1, \dots, \sigma_n) \in E_2^n$ и $|\sigma| = i$, называется (функциональным) *дешифратором* порядка n . Это связано с тем, что на любом наборе $\alpha, \alpha \in E_2^n$, значений переменных $x_1, x_2, \dots, x_n$ ровно одна из ФАЛ системы Q_n — ФАЛ с номером $|\alpha|$ — обращается в 1.

Функция M_n от (входных) переменных $x_1, x_2, \dots, x_n, y_0, y_1, \dots, y_{2^n-1}$ такая, что при всех $\alpha, \alpha \in E_2^n$, и $\beta, \beta \in E_2^{2^n}$, имеет место равенство: $M_n(\alpha, \beta) = \beta[i]$, где $i = |\alpha|$, называется (функциональным) *мультиплексором* порядка n . Если рассматривать переменные $x_1, x_2, \dots, x_n$ как «адресные», а переменные $y_0, y_1, \dots, y_{2^n-1}$ — как «информационные», то значение мультиплексора M_n равно значению той его информационной переменной, номер которой поступил на его адресные переменные. Легко убедиться в том, что для ФАЛ M_n справедливо представление:

$$M_n(x_1, x_2, \dots, x_n, y_0, y_1, \dots, y_{2^n-1}) = \bigvee_{i=0}^{2^n-1} Q_n[i] \cdot y_i. \quad (2)$$

Будем называть *схемным дешифратором* или *мультиплексором* любую СФЭ, которая реализует соответствующую систему ФАЛ.

Обозначим через θ_n систему всех 2^{2^n} различных ФАЛ множества P_2^n , в которой столбец значений ФАЛ $\theta_n[i]$, $0 \leq i \leq 2^{2^n} - 1$, рассматриваемый как двоичный набор длины 2^n , имеет номер i . Систему θ_n будем называть *универсальной системой ФАЛ порядка n* , а любую СФЭ, которая ее реализует, — *универсальным многополюсником порядка n* .

При построении более сложных СФЭ из более простых мы будем опираться на понятие подсхемы и принцип эквивалентной замены. СФЭ Σ' называется *подсхемой* СФЭ Σ , если множество ее вершин (дуг) является подмножеством множества вершин (соответственно дуг) Σ , при этом все те вершины Σ' , из которых в Σ выходит хотя бы одна дуга, не принадлежащая Σ' , или которые являются выходами Σ , объявляются выходами Σ' . Предполагается, что все дуги в Σ' имеют те же номера, что и в Σ , а все вершины Σ' имеют те же ФС, что и в Σ . Принцип эквивалентной замены для СФЭ вытекает из определения их функционирования и состоит в том, что если подсхему Σ' СФЭ Σ заменить эквивалентной ей СФЭ Σ'' , то полученная в результате СФЭ Σ_1 будет эквивалентна СФЭ Σ . В связи с этим подсхемы, которые не имеют общих внутренних вершин, можно рассматривать как (многовыходные) макроэлементы.

Лемма 4. Сложность минимального схемного дешифратора порядка n в базисе B_0 равна $2^n + O(n \cdot 2^{n-2})$.

Доказательство. Нижняя оценка. Поскольку любой дешифратор порядка n при $n \geq 2$ реализует систему из 2^n ФАЛ, отличных от переменных, то в нем должно быть не менее 2^n вершин, отличных от входов. Следовательно, сложность любого дешифратора порядка n ($n \geq 2$) в классе СФЭ не меньше, чем 2^n .

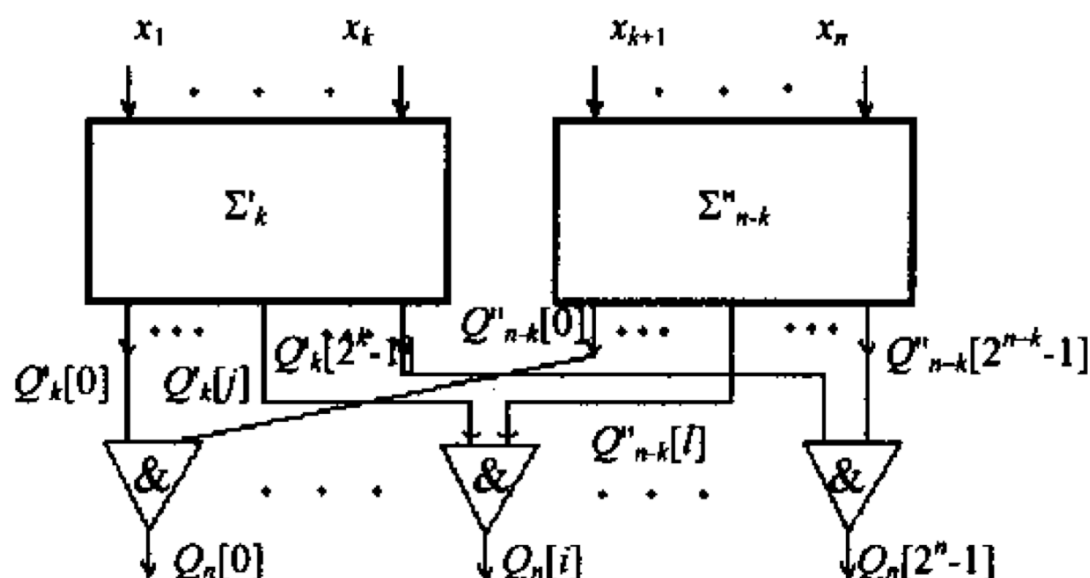


Рис. 2. Дешифратор порядка n .

Верхняя оценка. Для начала реализуем каждую функцию $Q_n[i]$ СФЭ в соответствии с формулой (1) ($i = 0, 1, 2, \dots, 2^n - 1$). Получим схемный дешифратор Σ'_n сложности $L(\Sigma'_n) \leq n \cdot 2^{n+1}$. Теперь построим более экономный дешифратор на основе имеющегося. Разобьем набор входных переменных $x = (x_1, x_2, \dots, x_n)$ на поднаборы $x' = (x_1, x_2, \dots, x_k)$ и $x'' = (x_{k+1}, \dots, x_n)$, где k — некоторый параметр и $1 \leq k \leq (n-1)$. Пусть Q' и Q'' — функциональные дешифраторы порядка k и $(n-k)$ от переменных x' и x'' , а Σ'_k и Σ'_{n-k} — соответствующие им схемные дешифраторы, построенные как описано выше. Легко видеть, что любую ФАЛ $Q_n[i]$, $0 \leq i \leq 2^n - 1$, можно представить в виде

$$Q_n[i] = Q'_k[j] \cdot Q''_{n-k}[l], \quad (3)$$

где $i = 2^{n-k}j + l$ и $0 \leq j \leq 2^k - 1$, $0 \leq l \leq 2^{n-k} - 1$. Дешифратор Σ_n порядка n от переменных x содержит дешифраторы Σ'_k , Σ'_{n-k} в качестве подсхем и реализует каждую ФАЛ $Q_n[i]$, $i = 0, 1, 2, \dots, 2^n - 1$, с помощью одного ФЭ конъюнкции, входы которого присоединены к выходам Σ'_k и Σ'_{n-k} (см. рис. 2) в соответствии с (3). Из построения Σ_n следует, что $L(\Sigma_n) = 2^n + L(\Sigma'_k) + L(\Sigma'_{n-k}) \leq 2^n + k \cdot 2^{k+1} + (n-k) \cdot 2^{n-k+1}$, и поэтому при $k = \lfloor n/2 \rfloor$ получим: $L(\Sigma_n) \leq 2^n + O(n \cdot 2^{n/2})$. Лемма доказана.

Лемма 5. Для любого натурального n существует схемный мультиплексор порядка n , который имеет сложность не более, чем $3 \cdot 2^n + O(n \cdot 2^{n/2})$.

Доказательство. Построим схемный мультиплексор S_n в соответствии с (2) на основе дешифратора Σ_n порядка n из леммы 4, который является его подсхемой. Для этого каждый выход Σ_n и соответствующий ему информационный вход S_n присоединим к входам ФЭ конъюнкции, а затем продизъюнктируем выходы всех таких ФЭ конъюнкции. Полученный таким образом мультиплексор S_n будет искомым.

Лемма 6. Минимальный универсальный многополюсник порядка n ($n \geq 1$) имеет сложность $2^{2^n} - n$.

Доказательство. Нижняя оценка следует из того, что

универсальный многополюсник порядка n реализует систему из $2^{2^n} - n$ ФАЛ, отличных от переменных. Для получения верхней оценки построим универсальный многополюсник Θ'_n по совершенным ДНФ всех реализуемых ненулевых ФАЛ (см. лемму 3), а затем перейдем от него к эквивалентной строго приведенной СФЭ Θ_n . Заметим, что число вершин СФЭ Θ_n , отличных от входов, не больше, чем число различных ФАЛ от переменных $x_1, x_2, \dots, x_n$, отличных от самих этих переменных. Следовательно, $L(\Theta_n) \leq 2^{2^n} - n$.

Перейдем к рассмотрению метода Шеннона для синтеза СФЭ. Идея метода заключается в построении СФЭ для произвольной функции n переменных с помощью мультиплексора порядка k и универсального многополюсника порядка $n-k$. Метод составляет содержание доказательства следующей теоремы.

Теорема. Для любой ФАЛ $f(x_1, x_2, \dots, x_n)$ можно построить СФЭ Σ_f , которая реализует f и для которой $L(\Sigma_f) \leq 6 \cdot \frac{2^n}{n} + O\left(\frac{2^n \cdot \log_2 n}{n^2}\right)$.

Доказательство. Разобьем набор входных переменных $x = (x_1, x_2, \dots, x_n)$ на поднаборы $x' = (x_1, x_2, \dots, x_k)$ и $x'' = (x_{k+1}, \dots, x_n)$, где k — некоторый параметр и $1 \leq k \leq n$. Для любого набора $\sigma' = (\sigma_1, \dots, \sigma_k)$ из E_2^k положим: $f_{[\sigma']} (x'') = f(\sigma', x'')$. Тогда для ФАЛ f будет справедливо представление:

$$\begin{aligned} f(x', x'') &= \bigvee_{\sigma' = (\sigma_1, \sigma_2, \dots, \sigma_k) \in E_2^k} x_1^{\sigma_1} x_2^{\sigma_2} \dots x_k^{\sigma_k} \cdot f_{[\sigma']} (x'') = \\ &= M_k(x', f_0(x''), f_1(x''), \dots, f_{2^k-1}(x'')) \end{aligned} \quad (4)$$

Построим СФЭ Σ_f из построенного по лемме 6 универсального многополюсника Θ_{n-k} порядка $(n-k)$ от переменных x'' и построенного по лемме 5 мультиплексора S_k порядка k с адресными переменными x' , на информационные входы которого подаются выходы Θ_{n-k} в

соответствии с (4). Тогда $L(\Sigma_f) = L(S_k) + L(\Theta_{n-k}) \leq 3 \cdot 2^k + O(k \cdot 2^{k^2}) + 2^{2^{n-k}}$. Полагая $k = \lfloor n - \log_2(n - 2 \log_2 n) \rfloor$, получим:

$$\begin{aligned} L(\Sigma_f) &\leq 6 \cdot \frac{2^n}{n - 2 \log_2 n} + O(n \cdot 2^{n/2}) + \frac{2^n}{n^2} = \\ &= 6 \cdot \frac{2^n}{n} + O\left(\frac{2^n \cdot \log_2 n}{n^2}\right). \end{aligned}$$

Следствие. $L(n) \leq 6 \cdot \frac{2^n}{n} + O\left(\frac{2^n \cdot \log_2 n}{n^2}\right).$

26. Вероятностное пространство. Случайные величины. Закон больших чисел в форме Чебышева.

ОПРЕДЕЛЕНИЕ 6.1. *Пространством элементарных исходов или событий называется любое непустое множество Ω .*

ОПРЕДЕЛЕНИЕ 6.2. *Алгеброй $\mathcal{A}$ называется класс (множество) подмножеств Ω , обладающий следующими свойствами (аксиомами):*

- 1) $\Omega \in \mathcal{A}$;
- 2) если $A \in \mathcal{A}$ и $B \in \mathcal{A}$, то $A \cup B \in \mathcal{A}$ и $A \cap B \in \mathcal{A}$;
- 3) если $A \in \mathcal{A}$, то $\bar{A} \in \mathcal{A}$.

ОПРЕДЕЛЕНИЕ 6.3. *σ -алгеброй $\mathcal{F}$ называется класс (множество) подмножеств Ω , обладающий следующими свойствами (аксиомами):*

- 1) $\Omega \in \mathcal{F}$;
- 2) если $A_1, A_2, \dots \in \mathcal{F}$, то $\bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$ и $\bigcap_{i=1}^{\infty} A_i \in \mathcal{F}$;
- 3) если $A \in \mathcal{F}$, то $\bar{A} \in \mathcal{F}$.

ОПРЕДЕЛЕНИЕ 6.4. *Событием или случайным событием называется элемент алгебры $\mathcal{A}$ или, в более общем случае, σ -алгебры $\mathcal{F}$.*

ОПРЕДЕЛЕНИЕ 6.5. *σ -алгебра подмножеств Ω называется минимальной, если она содержится в любой σ -алгебре подмножеств Ω . σ -алгебра подмножеств Ω называется максимальной, если она содержит любую σ -алгебру подмножеств Ω .*

ОПРЕДЕЛЕНИЕ 6.6. *Борелевской σ -алгеброй $\mathcal{B}$ называется σ -алгебра, порожденная множеством всех открытых интервалов.*

ОПРЕДЕЛЕНИЕ 6.7. *Наименьшая σ -алгебра, содержащая множество B , называется σ -алгеброй, порожденной B . Обозначение: $\sigma(B)$.*

ЗАМЕЧАНИЕ 6.4. *Под наименьшей σ -алгеброй (алгеброй), содержащей множество B , понимают σ -алгебру (алгебру), являющуюся пересечением всех σ -алгебр (алгебр), содержащих B .*

ОПРЕДЕЛЕНИЕ 6.8. *Элементы борелевской σ -алгебры называются борелевскими множествами.*

ОПРЕДЕЛЕНИЕ 6.9. *Пары $(\Omega, \mathcal{A})$ и $(\Omega, \mathcal{F})$ называются измеримыми пространствами. Элементы алгебры $\mathcal{A}$ и σ -алгебры $\mathcal{F}$ называются измеримыми множествами.*

(!) ОПРЕДЕЛЕНИЕ 6.10. Вероятностью или вероятностной мерой называется действительная функция случайного события $P : \mathcal{A} \rightarrow \mathbb{R}$, удовлетворяющая следующим аксиомам:

- 1) неотрицательность: $P(A) \geq 0, \forall A \in \mathcal{A}$;
- 2) нормированность: $P(\Omega) = 1$;
- 3) (конечная) аддитивность: если $A_1, \dots, A_n \in \mathcal{A}, A_i A_j = \emptyset, i \neq j$, то

$$P\left(\bigcup_{i=1}^n A_i\right) = \sum_{i=1}^n P(A_i).$$

ОПРЕДЕЛЕНИЕ 6.11. Вероятностью или вероятностной мерой называется действительная функция случайного события $P : \mathcal{F} \rightarrow \mathbb{R}$, удовлетворяющая следующим аксиомам:

- 1) неотрицательность: $P(A) \geq 0, \forall A \in \mathcal{F}$;
- 2) нормированность: $P(\Omega) = 1$;
- 3) σ -аддитивность (или счетная аддитивность): если $A_1, A_2, \dots \in \mathcal{F}, A_i A_j = \emptyset, i \neq j$, то

ОПРЕДЕЛЕНИЕ 6.12. Тройка $(\Omega, \mathcal{F}, P)$ называется вероятностным пространством. Тройка $(\Omega, \mathcal{A}, P)$ называется вероятностным пространством в широком смысле. Вероятность P также называется распределением вероятностей на Ω (на $(\Omega, \mathcal{F})$).

Вероятностью события A называется

$$P(A) = \lim_{N \rightarrow \infty} \frac{M}{N}$$

Такое определение вероятности вытекает из интуитивного представления. Математика имеет дело с абстрактными моделями, причем разные модели могут описывать одно и то же явление с различной степенью приближения и простоты. Приступим к построению такой модели для теории вероятностей.

Будем считать, что при осуществлении комплекса условий G могут происходить различные исходы опыта.

Множество всех простейших (неразложимых далее) взаимоисключающих исходов опыта назовем *пространством элементарных событий*. Обозначим множество элементарных

событий буквой Ω , а элементы этого множества – элементарные исходы - $\omega_1, \omega_2, \dots$.

Бросание монеты. При бросании монеты происходят два элементарных события: выпадение герба $\omega_1 (Г)$ и выпадение обратной стороны (цифры) $\omega_2 (Ц)$.

При двух бросаниях монеты имеется четыре элементарных исхода:

ω_1 - два раза выпал герб $(Г, Г)$;

ω_2 - первый раз выпал герб, а второй раз цифра $(Г, Ц)$;

ω_3 - первый раз выпала цифра, а второй раз герб $(Ц, Г)$;

ω_4 - оба раза выпала цифра $(Ц, Ц)$.

Перестановка различных предметов. Рассмотрим четыре предмета a, b, c, d . Пусть порядок их расположения является результатом какого-то опыта. Число элементарных событий равно $24=4!$ Возможные перестановки четырех предметов: $abcd, acdb, \dots, dcab, dcba$.

Возраст супругов. Страховые компании интересуются распределением возрастов супружеских пар. Пусть x - возраст мужа, y - возраст жены. Каждое наблюдение дает пару чисел (x, y) . За пространство элементарных событий следует принять первый квадрант $x > 0, y > 0$, так что каждая пара будет элементарным событием. Множество элементарных событий несчетно.

Итак, мы отправляемся от пространства элементарных событий и его точек; впредь они будут рассматриваться как данные, они являются первоначальными и неопределяемыми понятиями теории.

Рассмотрим событие A . При каждом элементарном исходе событие A может произойти или не произойти. Так в случае бросания двух монет событие A – герб выпал на 1-ом месте, происходит, если происходит элементарное событие ω_1 или ω_2 . В примере “возраст супругов” событие A – “жена младше мужа” происходит, если точка (x, y) лежит ниже прямой $y=x$.

Совокупность точек, при которой происходит событие A будем называть случайным событием A . Достоверным событием называется событие, состоящее из всех элементарных исходов, т.е. достоверное событие совпадает с Ω , невозможное событие не содержит элементарных исходов, т.е. это пустое множество $\emptyset$.

После введения понятия вероятностного пространства элементарные события рассматриваются как данные, первоначальные неопределяемые понятия теории, событие A рассматривается как элемент алгебры $\mathcal{A}$, а вероятность как функция $P(A)$, заданная на алгебре $\mathcal{A}$. Аксиоматика теории вероятностей была введена А.Н. Колмогоровым в 1939 г. В качестве примера вероятностного пространства рассмотрим классическую модель.

I. Пусть Ω состоит из конечного числа равновозможных элементарных исходов. Равновозможность событий $\omega_1, \omega_2, \dots, \omega_n$ означает, что с каждой точкой ω_i связано число, называемое вероятностью элементарного исхода и равное $\frac{1}{n} = p(\omega_i)$,

$$\sum_{i=1}^n p(\omega_i) = 1.$$

II. $\mathcal{A}$ – алгебра событий, состоит из всех подмножеств Ω .

III. Вероятность события A $P(A) = \frac{m}{n}$, где m – число элементарных исходов, входящих в A , а n – число всех элементарных исходов.

Заметим, что $P(A)$ равно сумме вероятностей элементарных исходов, входящих в A :

$$P(A) = \sum_{\omega_i \in A} p(\omega_i).$$

Задача 1. Монета брошена 2 раза. Найти вероятность того, что герб выпал на первом месте. Как мы уже отмечали, пространство элементарных исходов состоит из четырех событий: ГГ, ГЦ, ЦГ, ЦЦ. Будем считать, что вероятность каждого $\frac{1}{4}$. Тогда событие А “первым выпал герб” состоит из элементарных событий ГГ, ГЦ и

$$\text{вероятность } P(A) = \frac{1}{2} = \frac{2}{4}.$$

Геометрическая вероятность. Пусть на плоскости имеется некоторая область G и в ней содержится другая область g , области квадратуемы. В область G наудачу бросается точка. Выражению “точка бросается наудачу в область” придается смысл: брошенная точка может попасть в любую точку области G с одинаковой возможностью. Определим вероятность попадания в

$$\text{область } g \quad P = \frac{|g|}{|G|}, \text{ где } |G|, |g| - \text{площади областей } G \text{ и } g.$$

Таким образом мы определили следующее вероятностное пространство: $(\Omega, A, P(A))$.

I. Ω - множество всех точек области G . Оно несчетно.

II. A - множество всех квадратуемых областей $\{g\} \quad g \in G$. Это множество тоже несчетно. Событие A состоит в попадании точки в некоторую квадратуемую область $g(A) \in G$.

$$\text{III. } P(A) = \frac{|g(A)|}{|G|}$$

Свойства вероятности. Пользуясь аксиомами, определяющими вероятность, легко доказать следующие свойства вероятности.

1. Вероятность невозможного события равна 0.
2. Вероятность дополнения к событию $A \quad P(\bar{A}) = 1 - P(A)$.
3. Если событие A влечет за собой событие B , то $P(A) \leq P(B)$.
4. $P(A + B) = P(A) + P(B) - P(AB)$.

Утверждение 2.1. Вероятность невозможного события равна нулю.

$$P(\emptyset) \equiv 0.$$

Доказательство. Рассмотрим следующую счётную последовательность событий: $\Omega, \emptyset, \emptyset, \emptyset, \dots$. События из этой последовательности попарно несовместны и объединение их равно Ω . Далее пользуемся свойством счётной аддитивности для этой последовательности:

$$\Omega = \Omega \cup \emptyset \cup \emptyset \cup \emptyset \cup \dots \implies P(\Omega) = P(\Omega) + P(\emptyset) + P(\emptyset) + P(\emptyset) + \dots$$

$$\Downarrow$$

$$1 = 1 + P(\emptyset) + P(\emptyset) + \dots \implies 0 = P(\emptyset) + P(\emptyset) + P(\emptyset) + \dots \implies P(\emptyset) = 0. \quad \square$$

3°. Вероятность отрицания события — это единица минус вероятность самого события:

$$P(\bar{A}) \equiv 1 - P(A).$$

Доказательство.

$$\Omega = A \cup \bar{A}, A \cap \bar{A} = \emptyset \xrightarrow{2^\circ} P(A) + P(\bar{A}) = P(\Omega) = 1$$

$$\Downarrow$$

$$P(\bar{A}) = 1 - P(A). \quad \square$$

4° (Теорема сложения). Пусть A и B — произвольные события, тогда

$$P(A \cup B) \equiv P(A) + P(B) - P(AB).$$

Доказательство. Разложим A и B следующим образом: $A = AB \cup A\bar{B}$, события AB и $A\bar{B}$ несовместны; $B = AB \cup \bar{A}B$, события AB и $\bar{A}B$ — тоже несовместны. Несовместны и $A\bar{B}$ с $\bar{A}B$. Поэтому верно следующее:

$$P(A) = P(AB) + P(A\bar{B}), P(B) = P(AB) + P(\bar{A}B). \quad (C)$$

$$\text{Заметим, что } A \cup B = AB \cup A\bar{B} \cup \bar{A}B$$

$$\Downarrow$$

$$\begin{aligned} P(A \cup B) &= P(AB) + P(A\bar{B}) + P(\bar{A}B) \stackrel{C}{=} P(AB) + P(B) - P(AB) + P(A) - P(AB) \\ &= P(A) + P(B) - P(AB). \end{aligned} \quad \square$$

6° (Монотонность вероятности). Пусть A и B — некоторые события, причём B содержит в себе A . Тогда вероятность A не превышает вероятности B :

$$A \subseteq B \implies P(A) \leq P(B).$$

Доказательство.

$$B \setminus A = B \cap \bar{A}, A \cap (B \cap \bar{A}) = A \cap \bar{A} \cap B = \emptyset \cap B = \emptyset, B = A \cup B \setminus A$$

$\Downarrow$

$$P(B) = P(A) + P(B \setminus A) \geq P(A) \text{ в силу неотрицательности вероятности. } \quad \square$$

Далее в выкладках если в пересечении или в объединении верхний предел меньше нижнего, то такое пересечение следует считать равным Ω , а объединение — равным $\emptyset$. То есть,

$$\bigcup_{i=3}^2 = \emptyset, \bigcap_{i=6}^5 = \Omega.$$

Условная вероятность и основные формулы вероятности.
В определении вероятности события лежит некоторая совокупность условий G . Если же кроме условий G налагается дополнительное условие, например, произошло событие B , то приходим к определению условной вероятности.

Определение. Условной вероятностью события A при условии, что произошло событие B , называется $P(A|B) = \frac{P(AB)}{P(B)}$.

Легко проверить, что условная вероятность $P(A|B)$ удовлетворяет аксиомам вероятности.

Из определения условной вероятности получаем формулу умножения

$$P(AB) = P(A|B)P(B).$$

В случае, когда условная вероятность равна безусловной, получим $P(AB) = P(A)P(B)$.

Определение. Два события A и B называются независимыми, если $P(AB) = P(A)P(B)$.

Определение. События $A_1, A_2, \dots, A_n$ называются взаимно независимыми, если при всех комбинациях индексов $1 \leq i < j < k < \dots \leq n$ справедливы следующие правила умножения:

$$P(A_i A_j) = P(A_i)P(A_j)$$

$$P(A_i A_j A_k) = P(A_i)P(A_j)P(A_k)$$

$$\dots\dots\dots P(A_1 A_2 \dots A_n) = P(A_1)P(A_2) \dots P(A_n)$$

Предположим теперь, что событие B может осуществляться с одним и только одним из n несовместных событий $A_1, A_2, \dots, A_n$.

Иными словами $B = \sum_{i=1}^n BA_i$. Ясно, что события BA_i и BA_j несовместны, если $i \neq j$. По теореме сложения имеем:

$$P(B) = \sum_{i=1}^n P(BA_i), \text{ используя формулу умножения, получим}$$

$$P(B) = \sum_{i=1}^n P(B|A_i)P(A_i)$$

Это равенство носит название формулы *полной вероятности*.

Рассмотрим вероятность события A_i при условии, что произошло событие B :

$$P(A_i|B) = \frac{P(A_i B)}{P(B)}. \text{ Используя формулу полной вероятности,}$$

получим *формулу Байеса*

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{i=1}^n P(B|A_i)P(A_i)}$$

2. Случайная величина.

Определение. Случайной величиной $X(\omega)$ называется числовая функция, заданная на Ω , такая, что, для любого борелевского множества B множество элементарных исходов, при которых $X(\omega) \in B$ является событием, т.е. $\{\omega : X(\omega) \in B\} \in \mathcal{A}$.

Это определение эквивалентно тому, что для любых $x \in \mathbb{R}$ определена вероятность события $P\{\omega : X(\omega) \leq x\}$.

Определение. Функцией распределения случайной величины $X(\omega)$ называется функция действительного переменного $F(x) = P\{\omega : X(\omega) \leq x\} = P(X(\omega) \leq x)$.

Отметим основные свойства функции распределения.

- 1) $0 \leq F(x) \leq 1$
- 2) $F(x)$ непрерывна слева и не убывает.
- 3) $\lim_{x \rightarrow -\infty} F(x) = 0, \lim_{x \rightarrow +\infty} F(x) = 1$

Случайная величина называется *дискретной*, если она принимает конечное или счетное число значений. Функция распределения дискретной случайной величины кусочно-постоянная.

Случайная величина называется *непрерывной*, если существует

функция $y=p(x)$ такая, что $F(x) = \int_{-\infty}^x p(t)dt$. Функция $y=p(x)$

называется плотностью распределения, $p(x) = F'(x)$, если существует производная.

Свойства плотности распределения вытекают из свойств функции распределения.

- 1) $p(x) \geq 0$

$$2) \int_{-\infty}^{\infty} p(x)dx = 1 \quad \int_{-\infty}^{+\infty} p(x)dx = 1$$

Числовыми характеристиками случайной величины являются математическое ожидание и дисперсия.

Определение. Математическим ожиданием случайной величины $X(\omega)$ называется интеграл Лебега

$$EX(\omega) = \int_{-\infty}^{+\infty} x dF(x),$$

если он существует.

В случае дискретной случайной величины интеграл Лебега принимает вид

$$EX(\omega) = \sum_{i=1}^{\infty} x_i P(X(\omega) = x_i), \text{ где } x_1, \dots, x_n - \text{значения}$$

случайной величины.

В случае непрерывной случайной величины интеграл Лебега совпадает с интегралом Римана

$$EX(\omega) = \int_{-\infty}^{+\infty} xp(x)dx.$$

Свойства математического ожидания.

1. Если $X(\omega) = \text{const} = c$, то $EX = c$
2. $E(c_1 X + c_2 Y) = c_1 EX + c_2 EY$, где X и Y – случайные величины.

Будем в дальнейшем вместо $X(\omega)$ писать просто X .

Определение. Дисперсией случайной величины $X(\omega)$ называется

$$DX = E(X - EX)^2$$

Свойства дисперсии.

1. $DX = EX^2 - (EX)^2$
2. Если $X=c$, то $Dc = 0$
3. $D(cX) = c^2 DX$

Определение. Случайные величины $X(\omega)$, $Y(\omega)$ называются независимыми, если для любых борелевских множеств B_1 и B_2

$$P(X(\omega) \in B_1, Y(\omega) \in B_2) = P(X(\omega) \in B_1)P(Y(\omega) \in B_2).$$

В частности, для любых действительных значений x и y выполняется соотношение $P(X \leq x, Y \leq y) = P(X \leq x)P(Y \leq y)$, или функция совместного распределения случайных величин X и Y равна произведению функций распределения:

$$F_{x,y}(x,y) = F_x(x)F_y(y)$$

Для независимых случайных величин X и Y $E(XY) = EX \cdot EY$
и $D(X+Y) = DX + DY$.

Доказать самостоятельно.

3. Неравенство Чебышева. Закон больших чисел.

Математическое ожидание и дисперсию связывает соотношение, называемое неравенством Чебышева.

Теорема (неравенство Чебышева). Задана случайная величина X , у которой существует математическое ожидание и дисперсия, тогда для любого $\varepsilon > 0$

$$P(|X| \geq \varepsilon) \leq \frac{E|X|}{\varepsilon} \text{ и } P(|X - EX| \geq \varepsilon) \leq \frac{DX}{\varepsilon^2}.$$

Доказательство. Рассмотрим доказательство для дискретной случайной величины. По определению $E|X| =$

$$\begin{aligned} & \sum_{i=1}^{\infty} |x_i| P(X = x_i) = \\ &= \sum_{\omega: |x_i| \geq \varepsilon} |x_i| P(X = x_i) + \sum_{\omega: |x_i| < \varepsilon} |x_i| P(X = x_i) \geq \\ &\geq \sum_{\omega: |x_i| \geq \varepsilon} |x_i| P(X = x_i) \geq \varepsilon \sum_{\omega: |x_i| \geq \varepsilon} P(X = x_i) = \varepsilon P(|X| \geq \varepsilon). \end{aligned}$$

Поделим обе части на ε и получим первое неравенство. Если мы вместо случайной величины X в первом неравенстве подставим случайную величину $X - EX$, то получим

$$P(|X - EX| \geq \varepsilon) = P((X - EX)^2 \geq \varepsilon^2) \leq \frac{E(X - EX)^2}{\varepsilon^2} = \frac{DX}{\varepsilon^2},$$

т.е. мы доказали справедливость второго неравенства.

Теорема. Закон больших чисел. Если $X_1, \dots, X_n, \dots$ независимы и существует такая константа c , что $DX_i \leq c$, $i = 1, 2, \dots$, то при любом $\varepsilon > 0$

$$\lim_{n \rightarrow \infty} P\left\{ \left| \frac{X_1 + \dots + X_n}{n} - \frac{EX_1 + \dots + EX_n}{n} \right| < \varepsilon \right\} = 1$$

Доказательство. Обозначим $Y_n = \frac{X_1 + \dots + X_n}{n}$. Легко
 вычислить, что $EY_n = \frac{EX_1 + \dots + EX_n}{n}$, а

$$DY_n = \frac{DX_1 + \dots + DX_n}{n^2} \leq \frac{nc}{n^2} = \frac{c}{n}.$$

$$\begin{aligned} & P \left\{ \left| \frac{X_1 + \dots + X_n}{n} - \frac{EX_1 + \dots + EX_n}{n} \right| < \varepsilon \right\} \\ &= P \{ |Y_n - EY_n| < \varepsilon \} = \\ &= 1 - P(|Y_n - EY_n| \geq \varepsilon) \geq 1 - \frac{DY_n}{\varepsilon^2} \geq 1 - \frac{c}{\varepsilon^2 n}, \text{ откуда следует} \\ &\text{утверждение теоремы.} \end{aligned}$$

Для иллюстрации применения неравенства Чебышева рассмотрим следующий пример.

Вычислить с какой надежностью (вероятностью) среднее арифметическое измерений дает измеряемую величину α , если сделано 500 измерений с точностью 0,1 и дисперсия случайных величин (результатов измерений) не превосходит 0,3.

Итак, $X_1, \dots, X_n$ - результаты измерений,

$$EX_1 = \dots = EX_n = \alpha, \quad \varepsilon = 0,1, \quad n = 500.$$

$$\begin{aligned} & P \left(\left| \frac{X_1 + \dots + X_n}{500} - \alpha \right| \leq \varepsilon \right) = \\ & 1 - P \left(\left| \frac{X_1 + \dots + X_n}{500} - \alpha \right| \geq \varepsilon \right) \geq 1 - \frac{DX_1}{n\varepsilon^2} \\ &= 1 - \frac{0,3}{500 * (0,1)^2} = 0,94. \end{aligned}$$

Дискретные распределения.

1. Распределение Бернулли $B(1, p)$.

Случайная величина X принимает два значения 0 и 1
 $P(X = 0) = 1 - p, P(X = 1) = p, EX = p, DX = p(1 - p)$

Примером случайной величины с таким распределением служит случайная величина, принимающая значение 1, если произошло событие A , и 0, если оно не произошло.

2. Биномиальное распределение $B(n, p)$.

Случайная величина X принимает все значения от 0 до n .

$$P(X = k) = C_n^k p^k (1 - p), \quad \text{где} \quad 0 \leq k \leq n, \quad 0 < p < 1, \\ EX = np, DX = np(1 - p).$$

Примером случайной величины, имеющей такое распределение, служит случайная величина, принимающая значение k , если в n – независимых испытаниях с двумя исходами (успех, неуспех) произошло k успехов. Считается, что вероятность успеха не меняется от испытания к испытанию.

3. Распределение Пуассона $P(\lambda)$

Случайная величина X принимает значения 0, 1, 2...

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}, \quad \text{где } k \geq 0; \lambda = \text{const}; EX = \lambda, DX = \lambda.$$

Примером случайной величины, имеющей такое распределение, является случайная величина, равная числу вызовов поступивших на телефонную станцию в единицу времени.

Непрерывные распределения задаются плотностями распределения.

1. Равномерное распределение $R(a, b)$ с параметрами $-\infty < a < b < \infty$. Плотность распределения определяется формулами:

$$p(x, a, b) = \frac{1}{b - a}, \quad \text{если } x \in [a, b], \text{ в противном случае } p(x, a, b) = 0$$

$$EX = \frac{a + b}{2}, \quad DX = \frac{(b - a)^2}{12}.$$

2. Показательное (экспоненциальное) распределение $E(a, b)$ с параметрами $a > 0, -\infty < b < \infty$. Плотность показательного распределения задается формулами:

$p(x, a, b) = ae^{-a(x-b)}, x \geq b; \quad p(x, a, b) = 0$ в противном случае.

$$EX = b + \frac{1}{a}, DX = \frac{1}{a^2}$$

Показательное распределение широко используется в задачах теории надежности и массового обслуживания. Длительность бесперебойной работы приборов, длительность обслуживания, время ожидания обслуживания, интервалы времени между последовательными поступлениями заявок являются случайными величинами, имеющими показательное распределение.

3. Нормальное распределение $N(a, \sigma^2)$ с параметрами $-\infty < a < \infty, \sigma > 0, \delta > 0$. Плотность нормального распределения

$$p(x, a, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(x-a)^2}, \quad -\infty < x < \infty.$$

$EX = a, DX = \sigma^2$. Если $a = 0, \sigma = 1$, то распределение называется стандартным нормальным распределением.

Очень часто ошибки измерений подчиняются нормальному закону распределения, при этом параметр a равен систематической ошибке измерений, а σ^2 характеризует точность измерительного прибора.

В теории вероятностей и ее приложениях нормальное распределение играет особую роль, поскольку является предельным для сумм большого числа независимых случайных величин.

Центральная предельная теорема

Теорема. Если случайные величины $X_1, X_2, \dots$, одинаково распределены и имеют конечные $EX_i = a$ и $DX_i = \sigma^2$, то

$$\lim_{n \rightarrow \infty} P\left\{\frac{X_1 + \dots + X_n - na}{\sigma\sqrt{n}} \leq x\right\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{t^2}{2}} dt.$$

27. Квадратурные формулы прямоугольников, трапеций и парабол.

1. Задача численного интегрирования состоит в нахождении приближённого значения определённого интеграла $\int_a^b f(x)dx$, где

f – заданная непрерывная функция, и в оценке точности этого приближения. Основная формула интегрального исчисления

$\int_a^b f(x)dx = F(b) - F(a)$ позволяет выразить интеграл через перво-

образную $F(x)$ функции $f(x)$, но она мало пригодна для практики, так как получить ответ в элементарных функциях удаётся редко, а дающие этот ответ формулы могут быть слишком громоздкими и неудобными с вычислительной точки зрения. Кроме того, значения подынтегральной функции f могут быть известны только в конечном числе заранее заданных точек x_i . Поэтому многие правила построения приближённых квадратур основаны на замене интегрируемой функции f на более простую функцию, близкую к f , легко интегрируемую точно и принимающую в узлах x_i те же значения $f(x_i)$. В качестве такой приближающей функции берут либо алгебраический многочлен от x , либо рациональную функцию, либо тригонометрический многочлен – в зависимости от задачи. Если отрезок интегрирования $[a, b]$ конечный, а функция f имеет высокую гладкость, то можно рассчитывать на хорошее её приближение многочленом не очень высокой степени. Если же функция f не является непрерывной, а имеет особенности, то невозможно хорошо приблизить её многочленом. Если отрезок интегрирования бесконечен, то интеграл по нему от любого не равного нулю многочлена будет бесконечным. Поэтому разумной является только постановка задачи численного интегрирования функции f из класса функций с

заранее заданными свойствами. Будем считать, что f — достаточно гладкая функция, а отрезок интегрирования конечный.

Методы построения квадратур различаются числом выбранных узлов x_i , способами выбора узлов и принципами, на основе которых по выбранным узлам строится приближающая f функция. Чаще всего вычисление приближённого значения интеграла от f сводят к вычислению линейной комбинации значений $f(x_i)$ в конечном числе точек отрезка $[a, b]$. Но в некоторых случаях в эту линейную комбинацию включают и значения производных функции f во всех или в некоторых из рассматриваемых точек. При этом требуют чтобы коэффициенты такой линейной комбинации и узлы x_i не зависели от выбора конкретной функции f из рассматриваемого класса функций. (В других задачах, напротив, требуется выбрать узлы x_i , исходя из известного поведения функции f .) Удобным аппаратом построения квадратур с заранее заданными узлами x_i является аппарат интерполирования. Пусть, например, заданы n различных узлов x_i на отрезке $[a, b]$. Заменяем f на интерполяционный многочлен Лагранжа степени $n-1$

$$L_n(x) = \sum_{i=1}^n f(x_i) \cdot \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}. \quad \text{Положим}$$

$$\int_a^b f(x) dx = \int_a^b L_n(x) dx + R(f), \quad \text{где} \quad R(f) = \int_a^b (f(x) - L_n(x)) dx -$$

остаточный член. Можно вместо интерполяционного многочлена Лагранжа взять интерполяционный многочлен Эрмита, и тогда в квадратуру войдут наряду со значениями функции $f(x_i)$ ещё и значения её производных.

2. Заменим функцию f на отрезке $[a, b]$ интерполяционным многочленом нулевой степени, построенным по значению функции в средней точке отрезка: $L_1(x) = f\left(\frac{a+b}{2}\right)$. Тогда

$$\int_a^b f(x) dx = (b-a) \cdot f\left(\frac{a+b}{2}\right) + R(f).$$

Геометрически это означает, что площадь криволинейной трапеции, выражаемую интегралом

$$\int_a^b f(x) dx,$$

мы заменяем на площадь прямоугольника с основанием

$$b-a \quad \text{и} \quad \text{высотой} \quad f\left(\frac{a+b}{2}\right).$$

Поэтому формулу

$$\int_a^b f(x) dx \approx (b-a) \cdot f\left(\frac{a+b}{2}\right)$$

называют формулой прямоугольника. Если $f(x)$ имеет на $[a, b]$ непрерывную производную второго порядка, то для остаточного члена $R(f)$ можно получить простое выражение. По формуле Тейлора

$$f(x) = f(x_1) + (x - x_1) \cdot f'(x_1) + \frac{(x - x_1)^2}{2} f''(\xi(x)), \quad \text{где}$$

$$x_1 = \frac{a+b}{2}, \quad \xi \in (a, b).$$

Интегрируя, получим

$$\int_a^b f(x) dx = (b-a)f(x_1) + \frac{1}{2} \int_a^b (x - x_1)^2 \cdot f''(\xi(x)) dx.$$

Если известно, что $|f''(x)| \leq M$ на $[a, b]$, то для остаточного члена полу-

чим оценку $|R(f)| \leq \frac{M}{2} \cdot \int_a^b (x - x_1)^2 dx = \frac{(b-a)^3}{24} \cdot M$. Заметим,

что при интегрировании формулы Тейлора член $\int_a^b (x - x_1) dx \cdot f'(x_1) = 0$; этим и объясняется выбор середины от-

резка в качестве единственного узла x_1 интерполирования. Кроме того, очевидно, что формула прямоугольника даст точное значение интеграла, если функция f является любым многочленом первой степени.

В целях повышения точности интегрирования отрезок $[a, b]$ делят на m равных частей длины $h = \frac{b-a}{m}$ и применяют

формулу прямоугольника к каждому частичному отрезку. Тогда

$$\int_a^b f(x) dx = h \left[f\left(a + \frac{h}{2}\right) + \dots + f\left(a + \frac{2m-1}{2}h\right) \right] + R(f),$$

где $|R(f)| \leq \frac{(b-a)^3}{24 \cdot m^2} \cdot M$. При выводе последнего неравенства

надо учесть, что если функция $f''(x)$ непрерывна на $[a, b]$, а $\xi_1, \dots, \xi_m$ — некоторые точки этого отрезка, то на $[a, b]$ найдётся точка η такая, что среднее арифметическое

$\frac{1}{m} [f''(\xi_1) + \dots + f''(\xi_m)]$ равно $f''(\eta)$. Это неравенство означа-

ет, что обобщённая формула прямоугольников имеет второй порядок точности относительно шага равномерной сетки: $R(f) = O(h^2)$.

Заменим функцию f на отрезке $[a, b]$ интерполяционным многочленом первой степени, построенным по значениям функции в точках $x_1 = a$ и $x_2 = b$. При этом кривую $y = f(x)$ мы заменяем её хордой, соединяющей конечные точки:

$L_2(x) = f(a)\frac{x-b}{a-b} + f(b)\frac{x-a}{b-a}$, а площадь криволинейной трапеции — площадью трапеции с вершинами $(a,0)$, $(b,0)$, $(b,f(b))$, $(a,f(a))$:

$$\int_a^b f(x)dx = \frac{b-a}{2} \cdot (f(a) + f(b)) + R(f). \text{ Если } f(x) \text{ имеет на}$$

$[a,b]$ непрерывную производную второго порядка, то можно получить оценку остаточного члена $|R(f)| \leq \frac{(b-a)^3}{12} \cdot M$, где M является мажорантой для $|f''(x)|$ на $[a,b]$. Эта оценка получается из вида погрешности интерполяции для имеющей непрерывную вто-

рую производную функции f :

$$f(x) - L_2(x) = r(x) = \frac{1}{2!}(x-a)(x-b) \cdot f^{(2)}(\xi), \text{ где } a \leq \xi \leq b.$$

$$\text{Тогда } \left| \int_a^b r(x)dx \right| \leq \frac{M}{2} \left| \int_a^b (x-a)(x-b)dx \right| = \frac{(b-a)^3}{12} M.$$

Погрешность формулы трапеции можно снизить, если отрезок $[a,b]$ сначала разбить на части и к каждой части в отдельности применить формулу трапеции. Например, если разбить отрезок

$[a,b]$ на m равных частей длины $h = \frac{b-a}{m}$, то

$$\int_a^b f(x)dx = \frac{h}{2} [f(a) + \dots + 2f(a + (m-1)h) + f(b)] + R(f), \text{ где}$$

$|R(f)| \leq \frac{(b-a)^3}{12m^2} \cdot M$. Это значит, что обобщенная формула трапеций имеет второй порядок точности относительно шага равномерной сетки: $R(f) = O(h^2)$.

Заменим функцию f на отрезке $[a, b]$ интерполяционным многочленом второй степени, построенным по узлам $x_1 = a$,

$x_2 = \frac{a+b}{2}$, $x_3 = b$. Такой многочлен имеет вид

$$L_3(x) = f(a) \frac{(x-x_2)(x-b)}{(a-x_2)(a-b)} + f(x_2) \frac{(x-a)(x-b)}{(x_2-a)(x_2-b)} + \\ + f(b) \frac{(x-a)(x-x_2)}{(b-a)(b-x_2)}.$$

Геометрически это означает, что мы проводим параболу через конечные и среднюю точки кривой $y = f(x)$. Вычислим

$$\int_a^b L_3(x) dx = \frac{b-a}{6} [f(a) + 4f(x_2) + f(b)].$$

Имеем формулу параболы (формулу Симпсона):

$\int_a^b f(x) dx = \frac{b-a}{6} \cdot (f(a) + 4f(x_2) + f(b)) + R(f)$. Она по построению является точной для любого многочлена второй степени.

Кроме того, очевидно, что для функции $f(x) = (x-x_2)^3$ имеем

$$\int_a^b f(x) dx = 0 \text{ и } (f(a) + 4f(x_2) + f(b)) = 0.$$

Поэтому формула

параболы точна для любого многочлена третьей степени. Для оценки погрешности формулы параболы рассмотрим многочлен третьей степени $H_3(x)$, удовлетворяющий условиям $H_3(a) = f(a)$,

$H_3(x_2) = f(x_2)$, $H'_3(x_2) = f'(x_2)$, $H_3(b) = f(b)$. Это многочлен Эрмита, который интерполирует функцию f в двух однократных узлах $x_1 = a$, $x_3 = b$ и по значениям f и f' в двойном узле $x_2 = \frac{a+b}{2}$: $f(x) = H_3(x) + r(x)$, где $r(x)$ – погрешность интерполяции. Если функция f имеет на $[a, b]$ непрерывную производную четвертого порядка, то

$$r(x) = \frac{1}{4!}(x-a)(x-x_2)^2(x-b) \cdot f^{(4)}(\xi), \text{ где } a \leq \xi \leq b. \text{ По-}$$

скольку для многочлена третьей степени $H_3(x)$ формула параболы точна, имеем

$$\int_a^b f(x) dx = \frac{b-a}{6} \cdot (f(a) + 4f(x_2) + f(b)) + \int_a^b r(x) dx.$$

Поэтому $R(f) = \frac{1}{24} \int_a^b (x-a)(x-x_2)^2(x-b) \cdot f^{(4)}(\xi) dx$. Так

как множитель $(x-a)(x-x_2)^2(x-b)$ не меняет знак на отрезке $[a, b]$, а $f^{(4)}$ является непрерывной на $[a, b]$ функцией, то на этом отрезке найдётся такая точка η , что

$$R(f) = -\frac{1}{90} \left(\frac{b-a}{2} \right)^5 \cdot f^{(4)}(\eta). \text{ Если на } [a, b] \text{ выполнено нера-}$$

$$\text{венство } |f^{(4)}(x)| \leq M, \text{ то } |R(f)| \leq \frac{(b-a)^5}{2880} M.$$

Формулу параболы можно применить не сразу ко всему отрезку, а к отдельным его частям. Если, например, разбить $[a, b]$ на $2m$ равных отрезков, то для каждого отрезка $[a + (k-1)h, a + (k+1)h]$, где $h = \frac{b-a}{2m}$, получим

$$\int_{a+(k-1)h}^{a+(k+1)h} f(x)dx = \frac{h}{3} [f_{k-1} + 4f_k + f_{k+1}] + R(f), \quad f_k = f(a + kh).$$

Тогда
$$\int_a^b f(x)dx = \frac{b-a}{6m} [f_0 + f_{2m} + 2(f_2 + f_4 + \dots + f_{2m-2}) + 4(f_1 + f_3 + \dots + f_{2m-1})] + R(f),$$

где $|R(f)| \leq \frac{(b-a)^5}{2880 \cdot m^4} M$. Это значит, что обобщённая формула парабол имеет четвёртый порядок точности относительно шага равномерной сетки: $R(f) = O(h^4)$.

Пример. $\int_0^1 \sqrt{x} dx = \frac{2}{3}$. У подынтегральной функции даже

первая производная не ограничена на $[0,1]$. Поэтому все приведённые выше оценки погрешности численного интегрирования неприменимы.

Пример. $\int_{-1}^2 x \cdot |x| dx = \frac{7}{3}$. Подынтегральная функция непре-

рывная и гладкая, но её вторая производная имеет разрыв при $x = 0$. Формула параболы даст точный ответ только если её применить отдельно к каждому отрезку непрерывности второй производной: $-1 \leq x \leq 0$, $0 \leq x \leq 2$.

3. Рассмотренные три квадратуры относятся к формулам Ньютона-Котеса. Они получаются с помощью интегрирования интерполяционного многочлена и допускают стандартную запись. Зададимся некоторыми числами $d_1, \dots, d_n \in [-1,1]$ и построим интерполяционный многочлен $L_n(x)$ степени $n-1$, совпадающий с

$f(x)$ в точках $x_i = \frac{b+a}{2} + \frac{b-a}{2} d_i$. Положим

$\int_a^b f(x)dx \approx \int_a^b L_n(x)dx$. Тогда $R(f) = \int_a^b (f(x) - L_n(x))dx$ можно

оценить при помощи оценки погрешности интерполирования. Если функция f имеет непрерывную производную порядка n , то

$$f(x) - L_n(x) = \frac{\omega_n(x)}{n!} f^{(n)}(\xi), \text{ где } \omega_n(x) = (x - x_1) \cdot \dots \cdot (x - x_n),$$

а $\min\{x_1, \dots, x_n, x\} \leq \xi \leq \max\{x_1, \dots, x_n, x\}$. Поэтому

$$|R(f)| \leq \max_{[a,b]} |f^{(n)}(x)| \cdot \int_a^b \frac{|\omega_n(x)|}{n!} dx. \text{ Последний интеграл удобно}$$

записать в виде $D(d_1, \dots, d_n) \cdot \left(\frac{b-a}{2}\right)^{n+1}$, где

$$D(d_1, \dots, d_n) = \int_{-1}^1 \frac{|\omega_n^0(t)|}{n!} dt, \quad \omega_n^0(t) = (t - d_1) \cdot \dots \cdot (t - d_n); \text{ для}$$

того надо выполнить замену переменной $x = \frac{b+a}{2} + \frac{b-a}{2}t$. Если

все d_i различны, то $L_n(x) = \sum_{i=1}^n f(x_i) \cdot \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$. Следова-

тельно, получим квадратурную формулу вида

$$\int_a^b f(x) dx = \frac{b-a}{2} \sum_{i=1}^n D_i \cdot f\left(\frac{b+a}{2} + \frac{b-a}{2} d_i\right) + R(f), \text{ где коэф-}$$

фициенты $D_i = \int_{-1}^1 \prod_{j \neq i} \frac{t - d_j}{d_i - d_j} dt$, а для остаточного члена имеем

$$\text{оценку } |R(f)| \leq D(d_1, \dots, d_n) \cdot \max_{[a,b]} |f^{(n)}(x)| \cdot \left(\frac{b-a}{2}\right)^{n+1}. \text{ Для}$$

квадратур с нечётным числом узлов эту оценку можно уточнить, выразив её через $f^{(n+1)}$ вместо $f^{(n)}$. Чтобы это сделать, надо заменить подынтегральную функцию интерполяционным многочле-

ном, имеющим точку $\frac{b+a}{2}$ двукратным узлом интерполирования.

Пример. $n = 1, d_1 = 0$. Тогда $D = \int_{-1}^1 |t| dt = 1,$

$D_1 = \int_{-1}^1 1 \cdot dt = 2$. Получили формулу прямоугольника

$$\int_a^b f(x) dx \approx (b-a) \cdot f\left(\frac{a+b}{2}\right), \quad |R(f)| \leq \max_{[a,b]} |f'(x)| \cdot \frac{(b-a)^2}{4}.$$

Ту же самую квадратуру получим, если выберем $n = 2$, $d_1 = d_2 = 0$, то есть заменим $f(x)$ на

$$H_1(x) = f\left(\frac{a+b}{2}\right) + f'\left(\frac{a+b}{2}\right) \cdot \left(x - \frac{a+b}{2}\right). \quad \text{Тогда}$$

$$D = \int_{-1}^1 \frac{t^2}{2} dt = \frac{1}{3}, \quad \int_a^b H_1(x) dx = \int_a^b L_1(x) dx = (b-a) \cdot f\left(\frac{a+b}{2}\right), \text{ и}$$

$$\text{имеем} \quad \int_a^b f(x) dx = (b-a) \cdot f\left(\frac{a+b}{2}\right) + R(f), \quad \text{где}$$

$$|R(f)| \leq \max_{[a,b]} |f''(x)| \cdot \frac{(b-a)^3}{24}.$$

Таким образом, формулу прямоугольника можно рассматривать как квадратуру с кратным узлом.

4. Если сама функция f или её производные имеют особенности, то их надо выделить, раскладывая f на два сомножителя: $f(x) = p(x) \cdot g(x)$, где $p(x)$ имеет такие же особенности, а

$g(x)$ – достаточно гладкая функция. Аналогичный приём применяют

и при вычислении $\int_a^{\infty} f(x)dx$. Он позволяет охарактеризовать

закон убывания f на бесконечности при помощи функции $p(x)$ и выделить функцию $g(x)$, допускающую хорошее приближение многочленами. Функция $p(x)$ называется весовой. Процедура построения квадратур на основе интерполирования функции $g(x)$

полностью сохраняется при вычислении $\int_a^b p(x) \cdot g(x)dx$.

5. Говорят, что квадратура имеет алгебраическую степень точности k , если она точна для любого многочлена степени k , но не точна для некоторого многочлена степени $k + 1$. Алгебраическая степень точности формулы прямоугольника равна 1, формулы трапеции – 1, формулы параболы – 3. Для квадратур с нечётным числом узлов существенную роль играет их симметрия.

Теорема. Чтобы построенная по n узлам $x_1, \dots, x_n$ квадратурная

формула $\int_a^b f(x)dx \approx \frac{b-a}{2} \cdot \sum_{i=1}^n D_i \cdot f(x_i)$ была точной для всех алгеб-

раических многочленов степени $n - 1$, необходимо и достаточно, чтобы она была интерполяционной.

Дальнейшее повышение алгебраической степени точности требует отказа от априорного выбора равноотстоящих узлов квадратуры.

Теорема. Чтобы построенная по n узлам квадратурная формула была точной для всех алгебраических многочленов степени $2n - 1$, необходимо и достаточно, чтобы она была интерполяционной, а её узлы x_i , $i = 1, \dots, n$, были бы нулями многочлена $\omega(x) = (x - x_1) \cdot \dots \cdot (x - x_n)$, ортогонального ко всяко-

му многочлену $P_k(x)$ степени $k < n$: $\int_a^b \omega(x) \cdot P_k(x)dx = 0$.

Такие квадратуры называются квадратурами Гаусса. Их коэффициенты

D_i и числа $d_i = \frac{2x_i - (a+b)}{b-a}$ не зависят от отрезка $[a, b]$.

