Вычислительные системы, сети и телекоммуникации

Лекции – 17 часов

Лабораторные – 34 часа

Самостоятельная работа – 34 часа

Итоговая аттестация - экзамен

Общие принципы построения и архитектуры ЭВМ.

Краткая историческая справка.

Компьютер фон Неймана.

Потоки информации в ЭВМ.

Данные, типы элементарных данных.
Структуры данных и структурированная память.

Виды памяти и ее характеристики.

Процессор.

Мультипроцессорность.

Мультизадачность, разделение времени.
Реальное время.

Операционные системы.

УЧЕБНЫЕ ПОСОБИЯ

1. **Попов Ф.А. Вычислительные машины: общие принципы построения и архитектуры: Учебное пособие по части курса «Вычислительные системы, сети и телекоммуникации» для студентов специальности 080801. Алт. гос. техн. ун-т., БТИ. – Бийск: Изд-во Алт. гос. техн. ун-та, 2006. – 151 с.**
2. **Попов Ф.А., Гондурова Ю.В. Локальные вычислительные сети: Учебное пособие по части курса «Вычислительные машины, сети и системы телекоммуникаций» для студентов специальности 351400.Алт. гос. техн. ун-т., БТИ. – Бийск:Изд-во Алт. гос. техн. ун-та, 2005. – 83 с.**

КРАТКАЯ

ИСТОРИЧЕСКАЯ

СПРАВКА

Периоды создания и развития вычислительной техники

В истории развития вычислительных машин различают

домеханический,

механический, электрический и электронный периоды.

Домеханический период

Знаменательным событием в области усовершенствования инструментального счета было изобретение логарифмов. В 1614 г. шотландский математик Джон Непер (1550−1617 гг.) опубликовал трактат «Описание удивительных таблиц логарифмов» - первое руководство по вычислениям с помощью логарифмов.

Открытие логарифмов послужило основой для создания логарифмической линейки, появление которой относят к началу XVII века. Первые логарифмические линейки были изобретены в Англии. Почти 3,5 столетия логарифмическая линейка господствовала среди всех счетных средств.

Механический период

Биографии механических вычислительных машин ведутся от машины восемнадцатилетнего французского математика и физика Блеза Паскаля (1623 - 1662 гг.). Первую модель вычислительной машины, которая могла выполнять арифметические операции сложения и вычитания, он создал в 1642 г. В 1645 г. арифметическая машина «Паскалина», или «Паскалево колесо», получает законченный вид. До настоящего времени сохранилось восемь его машин. Одна из них находится в Музее искусств и ремесел в Париже, где собрана полная коллекция математических инструментов.

Первая счетная машина, которая механически производила сложение, вычитание, умножение и деление была изобретена в 1670 г. великим немецким математиком, физиком, философом и изобретателем Готфридом Вильгельмом Лейбницем (1646 - 1716 гг.). Машина получила название ***арифмометр***, ее окончательный вариант был завершен в 1710 г.

В XIX веке было сделано много открытий в области физики, станкостроения и автоматизации производства, которые положили начало интенсивному развитию вычислительной техники. В 1801 - 1804 гг. французский изобретатель Ж.М. Жаккар впервые использовал перфокарты для управления автоматическим ткацким станком.

Самым значительным событием XIX века в области создания вычислительной техники стал проект *разностной машины* английского математика Чарльза Бэббиджа (1791 - 1871 гг.), впервые в истории высказавшего идею создания вычислительных машин с программным управлением. Работать над машиной Ч. Бэббидж начал в 1812 г., к 1822 г. он построил небольшую действующую разностную машину и рассчитал на ней таблицу квадратов. Но более совершенную машину изготовить не удалось, поскольку в то время развитие техники находилось на недостаточно высоком уровне.

Совершенствуя разностную машину, ученый увидел возможность создания нового устройства, способного выполнять сложные вычислительные алгоритмы. В 1833 г. он приступил к работе над машиной, которую назвал *аналитической*. Она должна была отличаться большей скоростью и иметь более простую конструкцию, чем разностная машина.

Аналитическая машина состояла из трех основных блоков: устройства для хранения чисел и системы, которая передает эти числа от одного узла машины к другому (***склад***); устройства, позволяющего выполнять арифметические операции (***фабрика***); устройства для *управления* последовательностью действий машины. В конструкцию аналитической машины входило также ***устройство для ввода*** **исходных данных** ***и печати*** полученных результатов, т. е. ввод-вывод. Предполагалось, что машина будет действовать по программе, которая задавала бы последовательность операций и последовательность передач чисел со склада на фабрику и обратно.

1841 г. занялась изучением аналитической машины Ч. Бэббиджа Ада Августа Байрон (1815-1852 гг.), по мужу Лавлейс. А. Лавлейс разработала первые *программы* для аналитической машины, заложив тем самым теоретические основы программирования. Она впервые ввела понятие *цикла операции.* Ей принадлежат некоторые термины, употребляемые программистами и сейчас, например, *рабочие ячейки*. В единственном своем труде – в «Комментариях» она высказала важную мысль о том, что аналитическая машина может решать такие задачи, которые из-за трудности вычислений невозможно решить вручную.

Так впервые машина была рассмотрена не только как механизм, заменяющий человека, но и как устройство, способное выполнить работу, превышающую возможности человека.

В наши дни А. А. Лавлейс по праву называют первым программистом в мире.

Крупнейший русский математик и механик П.Л. Чебышев создает в 1878 г. арифмометр, выполнявший суммирование и вычитание. В 1881 г. он изобрел приставку к своему прибору для умножения и деления.

В 1880 г. петербургский инженер В.Т. Однер создает в России усовершенствованный арифмометр; в 1890 г. он налаживает массовый выпуск арифмометров, нашедших применение во всем мире. Данные в арифмометр вводились вручную, а привод осуществлялся вращением рукоятки. Простота работы с арифмометрами и надежность сделали их популярными. Их модификация «Феликс» выпускалась в СССР до 50-х годов XX века.

Важный шаг на пути автоматизации вычислений был сделан американцем Германом Холлеритом (1860-1929 гг.), который изобрел электромеханические машины для вычислений с помощью перфокарт, получившие название *счетно-аналитических* машин.

Работая с 1882 г. в Массачусетском технологическом институте и затем в Бюро патентов США, он начал разрабатывать машины для механизации обработки данных переписи населения. В 1888 г. он создает особое устройство - *табулятор*, в котором информация, нанесенная на перфокарты, расшифровывается с использованием электрического тока.

Перфокарты на специальной машине могли сортироваться по выбранному признаку, числа, пробитые в перфокартах, могли суммироваться, а сумма - пробиваться в перфокарте или печататься. В целом система Г. Холлерита включала перфокарту, перфоратор, позволяющий пробивать отверстия в нужных местах перфокарты, сортировальную машину и табулятор.

В 1896 г. Г. Холлерит основал фирму по выпуску перфокарт и счетно-перфорационных машин (СПМ). В дальнейшем она была преобразована в известную фирму - производитель вычислительной техники - IBM.

В 1919 г. академик Н.Н. Павловский создал метод исследования при помощи аналого-математического моделирования и дал ему полное теоретическое обоснование. Он же успешно применил новое средство вычислительной техники - *аналоговую вычислительную машину* (АВМ), которая была создана для реализации разработанного метода. Этот метод успешно развивался и действовал до 60-х гг., когда он был заменен цифровым моделированием на ЭВМ. Т.о., развитие АВМ в 20-30-х гг. XX века обогнало развитие цифровой техники, так как в этот период еще не было технологической базы, необходимой для создания универсальных ЭВМ. В области цифровой техники продолжала развиваться линия арифмометров и СПМ для выполнения учетных и статистических расчетных работ.

Одной из предпосылок создания ЭВМ было изобретение лампового диода и триода. В 1904 г. Дж.Флеминг (Великобритания) изобрел первый ламповый диод, а в 1906 г. Ли де Форест и Р. Либен (США) - первый триод. Но эра ЭВМ начинается с изобретения лампового триггера. В 1918 г. русский ученый М.А. Бонч-Бруевич изобрел триггер, имевший только два устойчивых положения равновесия: «открыто», «закрыто». Это изобретение имело большое значение для создания в дальнейшем современных вычислительных машин. В 1919 г. независимо от М.А. Бонч-Бруевича такой же прибор изобрели американцы У. Икклз (Экклз) и Ф. Джордан. Триггерные схемы постепенно стали широко применяться в электронике для переключения и релейной коммутации.

В 1931 г. французский инженер Р.-Л.В. Валтат выдвинул идею использования двоичной системы счисления при создании механических счетных устройств.

В начале XX века были проведены исследования в области полупроводников и сконструирована первая электронно-лучевая трубка. В 1907 г. русский ученый Б.Л. Розинг заявил патент на использование в телевидении электроннолучевой трубки. К середине 30-х гг. XX столетия в результате разработок В.К. Зворыкина и Ф. Франсуорта в США, К. Свинтона в Великобритании, В.П. Грабовского, Б.Л. Розинга, П.В. Тимофеева и др. в СССР появляются первые системы электронного телевидения.

В 1936 г. английский математик Алан Тьюринг (1912 - 1954 гг.) и независимо от него американский математик Э.Л. Пост (1879 - 1954 гг.) выдвинули и разработали концепцию *абстрактной вычислительной машины.*

А.Тьюринг опубликовал в 1936 г. статью в журнале Лондонского математического общества с доказательством того, что любой алгоритм может быть реализован с помощью дискретного автомата. Он предложил схему такого автомата, получившего название *машины Тьюринга*.

Машина Тьюринга - гипотетический универсальный преобразователь дискретной информации, теоретическая вычислительная система. Тьюринг и Пост показали принципиальную возможность решения автоматами любой проблемы при условии возможности ее алгоритмизации с учетом выполняемых им операций. Этими работами теоретически была доказана возможность создания универсальной цифровой вычислительной машины (ЦВМ). Перед второй мировой войной А. Тьюринг начал разрабатывать вычислительную машину с широкими логическими возможностями. За три года он разработал первый проект *электронного мозга* - автоматической вычислительной машины ACE - и первым подготовил ряд программ. В 1947 г. он занимается изучением проблемы обучения вычислительной машины.

В 1936 г. немецкий инженер-кибернетик Конрад Зюс начал работы по созданию универсальных автоматических цифровых машин с программным управлением на механических элементах. Это была последняя разработка, относящаяся к механическому периоду в истории развития вычислительных машин.

Электрический период

К 30-м годам XX века стала очевидной связь между релейными схемами и *алгеброй логики* (*булевой алгеброй*), основы которой заложил английский математик и логик Джордж Буль (1815 - 1864 г.г.) в работе 1847 г. «Математический анализ логики». Когда появилась принципиальная возможность создания средств вычислительной техники на электрической базе, логические операции, введенные Дж. Булем, оказались весьма полезны. Они изначально ориентированы на работу только с двумя сущностями: *истина* и *ложь*. Ясно, что они пригодились для работы с двоичным кодом, который в вычислительных машинах представляется двумя сигналами: *выключено* и *включено* (ноль и единица). Начиная с 30-х гг. XX века появляются вычислительные машины, использующие логические схемы для электромагнитных реле и оперирующие перфокартами. Эти машины могли выполнять довольно сложные арифметические вычисления.

Первая удачная попытка построить универсальную цифровую машину была предпринята в 1937 г. в США математиком Говардом Айткеном. Г. Айткен начал придумывать различные машины для автоматического решения частных задач, но затем пришел к идее автоматической универсальной вычислительной машины. Эта машина получила название *вычислительной машины с автоматическим управлением последовательностью операций* и известна под именем «Марк-1». Над первым вариантом машины Г. Айткен работал до 1944 г., машина создавалась на базе фирмы IBM и имела программное управление, программа набиралась на коммутационных досках и переключателях. Машина была выполнена на релейных и механических элементах.

Это еще не была машина с хранимой программой, однако она уже показала возможность построения автоматических вычислительных машин, состоящих из большого числа логических элементов. Основным логическим элементом в схемах, как и в СПМ, были реле. По сравнению с СПМ машина «Марк-1» имела достаточно длинную последовательность программных кодов и хорошее для своего времени быстродействие. Но, как и всякое механическое устройство, машина не обладала тем быстродействием, которое позволило бы осуществить качественный скачок в технологии вычислений. Улучшенная конструкция на реле повышенной надежности легла в основу цифровой вычислительной машины «Марк-2».

В конце 30-х гг. С.А. Лебедев (1902 - 1974 гг.) в Институте электротехники АН УССР приступил к конструированию ЭВМ, работающей в двоичной системе счисления. В 1941 г. работа была прервана.

В 1939 г. в США Дж. Стибниц закончил работу над релейной машиной фирмы «Белл», начатую в 1937 г. Машина выполняла арифметические операции над комплексными числами в двоично-пятеричной системе их представления. Это был релейный интерпретатор, управляемый программной перфолентой. В 1940 г. был проведен эксперимент по управлению на расстоянии вычислительной машиной «Белл-1». А в 1942 г. Дж. Стибниц сконструировал вычислительное устройство с программным управлением «Белл-2».

В 1940 г. в США под руководством Джона (Яноша) фон Неймана (1903 - 1957 гг.) разработан компьютер MANIAC (Mathematical Analyzer Numerical and Computer).

К первым универсальным ЦВМ с программным управлениям на электромеханических элементах относят также машины, разработанные в Германии К.Зюсом к 1941 г. – «Зюс -2» и «Зюс-3». Машина «Зюс-3» была релейной, для нее был разработан язык программирования, она использовалась при расчетах ракет.

Одной из наиболее совершенных релейных вычислительных машин была советская машина РВМ-1, сконструированная в начале 50-х г.г. выдающимся инженером Н.И.Бессоновым (1906 - 1963 гг.) и построенная в 1956 г. Эта машина успешно работала до 1966 г.

Главными недостатками релейных машин являлось отсутствие хранимой программы, что обусловливалось небольшим объемом оперативной памяти, и невысокая скорость работы, вызванная низким быстродействием электромеханических релейных переключателей.

Электронный период

В начале 40-х годов XX века потребность в автоматизации вычислений стала настолько велика, что над созданием машин типа построенных К. Зюсом и Г. Айткеном одновременно работало несколько групп исследователей. В США исследования с ЭВМ продолжил Джон Моучли, который в 1942 г. предложил проект вычислительной машины, предназначенной для военных целей. В 1943 г. начались работы над реализацией проекта Дж. Моучли в Пенсильванском университете под руководством Дж. Моучли и инженера-электронщика Д.П. Эккерта. Ученые начали конструировать вычислительную машину на основе электронных ламп, а не электрических реле. Их машина, названная ENIAC (ЭНИАК) - «Electronic Numerical Integrator and Computer» («электронный цифровой компьютер»), в основном была закончена в 1944 г. Окончательный вариант ЭНИАК, долгое время считавшейся первой электронно-вычислительной машиной, был введен в строй 15 февраля 1946 г.

ЭНИАК предназначался для использования при расчетах баллистических таблиц для орудий береговой обороны США. Первым найденным в США практическим применением ЭНИАК было решение задач для проекта атомной бомбы. В последующее время ЭНИАК использовался в Америке в основном в военных целях: для составления артиллерийских таблиц и таблиц прицельного сбрасывания бомб с самолетов.

Одновременно с постройкой ЭНИАК создавалась ЭВМ и в Великобритании, с целью дешифровки кодов, которыми пользовались вооруженные сила Германии в период второй мировой войны. Математический метод дешифровки был разработан группой математиков, в число которых входил А. Тьюринг. В начале 40-х годов А. Тьюринг совместно с Х.А. Ньюменом сконструировали машину «Colossus-1», которая в 1943 г. начала работать и которую можно считать первым *электронным компьютером.* Хотя и ЭНИАК и «Колосс» работали на электронных лампах, они по существу копировали электромеханические машины: новое содержание (электроника) было втиснуто в старую форму (структуру доэлектронных машин).

В 1945 г. Дж. фон Нейман разработал концепцию электронно-вычислительной машины EDVAC (Electronic Discrete Variable Computer) с вводимыми в память программами и числами. Сама машина была завершена в 1950 г. Главными элементами концепции были: принцип хранимой программы и принцип параллельной организации вычислений, согласно которому операции над числом проводятся по всем его разрядам одновременно. Запущена машина в эксплуатацию была лишь в 1952 г.

В 1946 г. он выпустил отчет «Предварительное обсуждение логической конструкции электронной вычислительной машины», в котором и были изложены основные принципы логической структуры ЭВМ нового типа.

В отчете обоснованы эти принципы и даны технические проработки их реализации. В нем утверждается, что ЭВМ должна создаваться на электронной основе и работать в **двоичной системе счисления.** В ее состав должны входить: ***арифметическое устройство*** с представлением чисел в форме с плавающей точкой, центральное ***устройство управления***, ***запоминающие устройства*** (в т.ч. оперативное запоминающее устройство для чисел и команд и связанное с ним внешнее запоминающее устройство большой емкости), **устройства *ввода данных* и *вывода* результатов на печать.** В системе команд должны быть команды ***условной* и *безусловной* передачи управления.**

Первый компьютер, в котором были воплощены принципы фон Неймана, был построен в 1949 г. английским профессором Морисом В. Уилксом в Кембриджском университете. Машина называлась EDSAC (Electronic Data Storage Automatic Computer).

Машина «Джониак», построенная в 1946-1952 годах и названная так в честь фон Неймана, имела память на электроннолучевой трубке специальной конструкции. Позднее здесь был применен для запоминающего устройства магнитный барабан, впервые использованный в 1947 г. в небольшой английской вычислительной машине, построенной под руководством Э. и К. Бут. Магнитная лента была впервые использована в 1951 г. на машине UNIVAC-1, построенной Дж. Моучли и Д.П.Эккертом. Это была первая большая вычислительная машина, построенная не по специальному заказу, а для продажи.

Несмотря на то, что исследования в области электроники в нашей стране были начаты на несколько лет позже, чем в США и Великобритании, в сжатые сроки был выполнен ряд проектов цифровых ЭВМ. Первые ЭВМ в нашей стране создавались для решения сложных и трудоемких математических задач. Работы над первой ЭВМ начались в 1946 г. под руководством академика Сергея Алексеевича Лебедева в Институте электротехники АН УССР. В 1947 г. был закончен проект малой электронной счетной машины - МЭСМ. Основные опытно-конструкторские работы, монтаж и испытания прошли в 1949 -1950 гг. В октябре 1951 г. машина была введена в эксплуатацию. **Это была самая быстродействующая тогда машина в Европе.** С ее помощью был решен ряд важных задач, в том числе расчет устойчивости магистральной линии электропередачи Куйбышев - Москва (1951 г.). **Функциональная схема машины удовлетворяла всем принципам Дж. фон Неймана.**

В 1952 г. вступил в строй опытный образец машины БЭСМ (большая электронно-счетная машина), созданный в Институте точной механики и вычислительной техники под руководством С.А.Лебедева. По многим параметрам БЭСМ превосходила EDVAC, в ней были осуществлены решения, вошедшие в практику вычислительной техники только через несколько лет. БЭСМ-2 имела оперативную память на электронно-акустических линиях задержки, затем - на электронно-лучевых трубках и позже - на ферритовых сердечниках. Внешнее запоминающее устройство состояло из магнитных барабанов и магнитной ленты.

В Лаборатории управляющих машин и систем Энергетического института АН СССР под руководством И.С. Брука и М.А. Карцева в 1952 г. была введена в действие машина М-2 средней мощности и малая машина М-3.

Серийное производство ЭВМ в нашей стране началось в 1953 г. Под руководством Ю.Я. Базилевского и Б.И. Рамеева была разработана машина «Стрела», выпускавшаяся малой серией. В 1955 г. в Пензе начался выпуск малой ЭВМ «Урал-1», созданной также под руководством Б.И. Рамеева. В 50-е годы также разработаны серийные ЭВМ: в Ереване под руководством Ф.Т. Саркисяна – «Раздан», в Минске (В.В. Пржиялковскй и др.) - ЭВМ «Минск», а в Киеве – «Киев». В 1958 г. самой мощной в Европе была советская ЭВМ М-20. Она была создана объединенными усилиями коллективов С.А. Лебедева и Ю.Я. Базилевского.

С переходом к серийному производству ламповых ЭВМ с хранимой программой начинается период машин *первого поколения*. Здесь история развития вычислительной техники переходит в историю поколений ЭВМ.

ПОКОЛЕНИЯ ЭВМ

Первое поколение

Первое поколение ЭВМ (с начала и до конца 50-х гг. XX века) - машины громоздкие, использовавшиеся в крупных научных центрах для решения задач в области ядерной физики, ракетной техники, космоса и метеорологии. Они отличались большим потреблением энергии, низкой надежностью, малым быстродействием. Программирование на них - в машинных кодах.

Нововведения: Улучшение характеристик электронных ламп и постепенный переход к полупроводникам завершился заменой ламповых триодов транзисторами. Первые серийные универсальные ЭВМ на транзисторах выпущены в 1958 г. в США, Японии и ФРГ, в 1959 г. - в Великобритании, в 1960 г. - во Франции, в 1961 г. - в СССР. Была создана оперативная память на ферритовых сердечниках и внешняя память на МД. В 1957 г. начался серийный выпуск машин с памятью на МД. В организации вычислительного процесса нововведение - *совмещение во времени вычислений и ввода-вывода информации (*серийные модели, разработанные в США (IBM-704 (1956 г.) и IBM-709 (1958 г.)) и в СССР (М-20 (1958 г.)). В программировании - разработки методов программирования в символических обозначениях и появление алгоритмических языков. В СССР в 1952-1953 гг. А.А. Ляпунов разработал операторный метод программирования, а в 1953-54 гг. Л.В.Канторович - концепцию крупноблочного программирования.

Т.о., с первым поколением ЭВМ связаны разработки основ программирования, достижение сравнительно емкой и быстрой памяти, переход к ЭВМ на стандартных и специализированных блоках, разработка устройств ввода-вывода (УВВ) информации на перфокартах и перфолентах.

Характерные черты ЭВМ первого поколения:

- Элементная база: электронно-вакуумные лампы, резисторы, конденсаторы. Соединение элементов - навесной монтаж проводами;
- Габариты: ЭВМ выполнена в виде громоздких шкафов и занимает специальный машинный зал;
- Быстродействие: 10-20 тыс. оп/сек;
- Эксплуатация: слишком сложна из-за частого выхода из строя;
- Программирование: трудоемкий процесс в машинных кодах. Общение с ЭВМ требовало от специалистов высокого профессионализма.

Второе поколение

Второе поколение - конец 50-х - начало 60-х годов XX века. Использование транзисторов позволило уменьшить размеры и потребляемую мощность устройств. Первый компьютер на транзисторах - в США в 1954-1957 гг. Всего в мире было изготовлено около 30 тыс. таких ЭВМ, причем произошел сдвиг в области их применения. Если в начале 50-х гг. ЭВМ использовались для научно-технических расчетов, то в 60-е гг. первое место стала занимать обработка символьной информации, в основном экономической. К середине 60-х годов появились более компактные внешние устройства для компьютеров, что позволило фирме Digital Equipment Corporation выпустить в 1965 г. первый мини-компьютер PDP-8.

Более дешевая элементная база позволила удешевить ЭВМ, уменьшить их габариты, повысить быстродействие и емкость памяти. Получает развитие мультипрограммирование - совмещение во времени выполнения различных программ, а также переход от машинных к алгоритмическим языкам. К 1967 г. - уже около 1000 таких языков. В 1953-57 гг. в США (Дж. Бейкус) разработан язык Fortran – «переводчик формул». В 1957 г. - алгоритмический язык Algol. В конце 50-х г. Дж. Маккарти в Массачусетском технологическом институте (МТИ, США) разрабатывает LISP для работ по проблеме искусственного интеллекта. В 1960 г. в США создается COBOL – язык для обработки коммерческой информации. В этом же г. С.Пейперт с коллегами из МТИ предлагают язык LOGO, с помощью которого можно управлять «черепахой» - программной моделью малого робота. В 1965 г. Дж.Кемени и Т.Курц в США разрабатывают язык BASIC, который первоначально предназначался для вводного курса по информатике. BASIC - Beginner's All-purpose Symbolic Instruction Code - многоцелевой символический код-инструкция для начинающих. ПО становится фактором, определяющим стоимость машины.

Первые ЦВМ на полупроводниках в СССР - с 1961 г. Для научных расчетов - ЭВМ средней мощности: М-220, БЭСМ-3, БЭСМ-4 (Москва), Урал-11, -14, -16 (Пенза), в Минск-22, -23, -32 (Минск), Раздан-2, -3 (Ереван). В 1961 г.- первая в стране серийная универсальная полупроводниковая ЭВМ Днепр-1. Принципы совместимости машин (однотипность архитектуры) и агрегатной комплектации был осуществлен на серии «Урал» раньше, чем он был реализован на IBM-360.

В ЭВМ Минск появилось алфавитно-цифровое печатающее устройство (АЦПУ). Это был громоздкий агрегат, который позволял печатать на перфорированной бумаге форматированный текст. В первой половине 70-х гг. самой распространенной машиной в СССР стала Минск-32, имевшая неплохую ОС, довольно мощные системы программирования, пишущую машинку в качестве терминала оператора.

В 1967-1969 гг. в Институте точной механики и вычислительной техники АН СССР под руководством С.А. Лебедева и В.А. Мельникова создаются первые полупроводниковые мини-ЭВМ для научных расчетов, под руководством Г.Е. Овсепяна в Ереване - Наири-2, под руководством В.М. Глушкова в Киеве (институт кибернетики АН УССР)- Мир-2. Для машин «Мир» создаются специальные входные языки Мир, Аналитик. В Мир-2 впервые диалог с пользователем осуществляется с помощью дисплея со световым пером. В это время в СССР развивается применение ЭВМ для управления технологическими процессами сбора и обработки экспериментальных данных в реальном масштабе времени, для планово-экономических расчетов. Сдана в эксплуатацию первая в стране АСУ предприятием с массовым производством – «Львов». В институте кибернетики АН УССР создается управляющая ЭВМ Днепр-2 с развитой системой прерывания, обеспечивающей одновременную работу с более 1600 входных и более 100 выходных аналоговых устройств различных классов.

Лучшей в мире ЭВМ второго поколения, уровень которой на несколько лет опередил уровень зарубежных аналогичных ЭВМ, стала БЭСМ-6, созданная в Институте точной механики и вычислительной техники АН СССР под руководством С.А. Лебедева и В.А. Мельникова. БЭСМ-6 работала в мультипрограммном режиме. Обладая высоким быстродействием (около 1 млн. в сек.), БЭСМ-6 по своей архитектуре была ближе к ЭВМ третьего поколения. До 1973 г. она была одной из наиболее производительных однопроцессорных машин в мире. Машина имела систему автоматизированного программирования с входными языками: Фортран, Алгол-60, ЛИСП, др. Программное обеспечение выполнялось Л.Н. Королевым, М.Р. Шура-Бурой, Н.Н. Говоруном, Э.З. Любимским и др. В дальнейшем на БЭСМ-6 используется разработанная в Новосибирске под руководством А.П.Ершова α-система программирования с расширением языка Алгол-60. Эта машина широко использовалась при разработке и реализации отечественных космических программ. БЭСМ-6 выпускалась серийно до 1981 г. и использовалась на ВЦ до конца 90-х г.г.

Период машин второго поколения характеризуется крупнейшими сдвигами в архитектуре ЭВМ, их ПО, организации взаимодействия человека с машиной. Эти достижения в дальнейшем применяются в машинах третьего и четвертого поколений. Это в первую очередь относится к многопроцессорным системам, организации мультипрограммной работы, разделению времени. Формируются основы процедурной организации взаимодействия человека с машиной.

Характерные черты ЭВМ второго поколения:

- элементная база: полупроводниковые элементы; соединение элементов – печатные платы и навесной монтаж;
- габариты: ЭВМ выполнены в виде однотипных стоек, для их размещения требуется специально оборудованный машинный зал, в котором под полом прокладываются кабели, соединяющие между собой многочисленные автономные устройства;
- производительность: до 1 млн оп/с.;
- эксплуатация: появились ВЦ с большим штатом обслуживающего персонала, где устанавливались обычно несколько ЭВМ. При выходе из строя нескольких элементов производилась замена целиком всей платы, а не каждого элемента в отдельности, как в ЭВМ предыдущего поколения;
- программирование: велось преимущественно на алгоритмических языках. Программисты передавали свои программы на перфокартах или магнитных лентах операторам ЭВМ на ВЦ для организации дальнейших расчетов на ЭВМ. Решение задач производилось в пакетном режиме, т. е. все программы вводились в ЭВМ подряд друг за другом, а их обработка велась по мере освобождения соответствующих ресурсов. Результаты решения распечатывались на специальной перфорированной бумаге.

Третье поколение

К 1964 г. появились ЭВМ *третьего поколения* на интегральных схемах (ИС). Период создания и внедрения ЭВМ третьего поколения - от начала 60-х до середины 70-х гг. Идея создания ИС была предложена в 1952 г. инженером из Великобритании Дж. Даммером. В 1959 г. Роберт Нойс (будущий основатель фирмы Intel) изобрел метод, позволивший создавать на одной пластине и транзисторы, и все необходимые соединения между ними. Полученные электронные схемы и стали называться интегральными схемами, или *чипами.*

В 1961 г. в продажу поступила первая, выполненная на пластине кремния ИС на 6 элементах, в 1963 г. ИС имела 10-20 элементов, в 1967 г. - примерно 100, к 1970 г. – 1000, к 1975 г. – 30000, к 1982 г. – 300000 элементов на кристалле в несколько квадратных миллиметров. В 1968 г. фирма Burroughs выпустила первый компьютер на ИС, а в 1970 г. фирма Intel начала продавать ИС памяти.

Использование ИС привело к снижению габаритов ЭВМ, повышению их надежности, увеличению производительности. В это время появились внешние запоминающие устройства на МД, обладающие емкостью в сотни миллионов бит и высоким быстродействием. Основным режимом эксплуатации машин этого поколения является режим мультипрограммирования, когда в ЭВМ одновременно обрабатывается несколько программ, находящихся на разных стадиях выполнения.

Начало периода машин третьего поколения связано с разработкой ЭВМ серии IBM-360, позднее – IBM-370 (IBM - International Business Machines Corporation), оказавшей огромное влияние на развитие вычислительной техники во всем мире. Выпуск машин IBM-360 был начат в США в 1965 г. Сущность идей, заложенных в проект IBM-360 - создание семейства машин на ИС, имеющих широкий диапазон производительности и совместимых на уровне машинных языков, периферийных устройств, модулей конструкции и системы элементов, а также операционных систем (ОС).

Значительным явлением в этот период было создание мини-ЭВМ. Сущность идеи состояла в такой минимизации аппаратуры центрального вычислителя, которая позволяла создать универсальные ЭВМ, способные осуществлять *управление в реальном масштабе времени.*

В этот период развитие ЭВМ в СССР проходило в направлении создания ЭВМ единой системы (ЕС ЭВМ), в основных чертах копирующих IBM-360 и IBM-370, программно-совместимых между собой, а также с ЭВМ типа IBM-360.

К третьему поколению ЭВМ принадлежат малые вычислительные машины СМ - СМ-1, СМ-2, СМ-3 и СМ-4. Наличие ЗУ различной емкости и быстродействия, стандартизация подключения к процессору устройств ввода-вывода символьной и графической информации открыли возможности использования СМ ЭВМ в различных измерительных, управляющих, информационных комплексах. Они сопрягались с ЕС ЭВМ как периферийные процессоры, удаленные терминалы, процессоры ввода-вывода, связи ЭВМ с объектом в реальном масштабе времени.

В это время в СССР выпускаются мини-ЭВМ Мир-31, Мир-32, Наири-34, ЭВМ серии АСВТ М-6000 и М-7000 для управления технологическими процессами; на ИС - настольные мини-ЭВМ М-180 «Электроника-100, -200», «Электроника ДЗ-28», «Электроника НЦ-60» и другие.

Вклад отечественной науки в мировое развитие электронной вычислительной техники в этот период связан с идеями и разработками машины М-10 (1975 г., главный конструктор М. А. Карцев). М-10 была первой в мире промышленно освоенной *многопроцессорной* ЭВМ.

В структуру машин третьего поколения вводятся ОС, облегчающие программирование, связь пользователя с машиной, ее обслуживание. В состав этих машин включали системы, работающие в режиме реального времени - встроенные микропроцессоры, позволявшие практически немедленно обрабатывать всю поступающую от других систем, периферийных устройств и датчиков информацию и тут же посылать обратно управляющие сигналы. Мини-ЭВМ стали использоваться в станках с числовым программным управлением, для управления роботами-манипуляторами и т.п.

Однако компьютеры третьего поколения оставались сложными в обслуживании. Большим ЭВМ требовались специальные помещения, а малые плохо подходили для установки на небольших движущихся объектах: самолетах, космических кораблях. Высокая стоимость не позволяла использовать эти компьютеры в недорогих технических устройствах.

Характерные черты ЭВМ третьего поколения:

- элементная база — ИС;
- габариты: для размещения ЕС ЭВМ требуется машинный зал. А малые ЭВМ – это, в основном, две стойки, дисплей. Они не нуждались, как ЕС ЭВМ, в специально оборудованном помещении;
- производительность: сотни тысяч — миллионы операций в секунду;
- эксплуатация: более оперативно производится ремонт стандартных неисправностей, но из-за большой сложности системной организации требуется штат высококвалифицированных специалистов. Незаменимую роль играет системный программист;
- технология программирования и решения задач: во многих ВЦ появились дисплейные залы, где каждый программист мог подсоединиться к ЭВМ в режиме разделения времени. Как и прежде, основным оставался режим пакетной обработки;
- произошли изменения в структуре ЭВМ. Наряду с микропрограммным способом управления используются принципы модульности и магистральности. **Модульность проявляется в построении компьютера на основе набора модулей – конструктивно и функционально законченных электронных блоков в стандартном исполнении. Под магистральностью понимается способ связи между модулями компьютера, когда все входные и выходные устройства подсоединены одними и теми же проводами (шинами). Это - прообраз современной системной шины;**
- увеличились объемы памяти. Появились дисплеи, графопостроители.

Четвертое поколение. Персональные компьютеры

Переход к машинам *четвертого поколения* – ЭВМ на больших интегральных схемах (БИС) происходил в середине и второй половине 70-х годов и завершился приблизительно к 1980 г. Машины этого поколения развивались, во-первых, в направлении создания мощных многопроцессорных систем, имеющих производительность до сотен миллионов операций в секунду, и, во-вторых, в направлении создания дешевых компактных мини- и микро-ЭВМ.

Большие ЭВМ четвертого поколения представляют собой многопользовательские машины с развитыми возможностями для работы с базами данных, с различными формами удаленного доступа. Они имеют несколько процессоров и ориентированы на выполнение определенных операций, процедур или на решение некоторых классов задач. Выпуск их стал увеличиваться, хотя их доля в общем парке постоянно снижается. Серия машин IBM S/390 продолжила линию машин IBM-360 и IBM-370. Они позволяют задавать переменную конфигурацию (число процессоров - 1-10, емкость оперативной памяти - 512-81292 Мбайта, число каналов - 3-256).

К числу машин четвертого поколения относятся и машины RS/6000, предназначенные для создания графических рабочих станций, Unix-серверов, кластерных комплексов. Первоначально эти машины предполагалось применять для обеспечения научных исследований.

К числу средних ЭВМ четвертого поколения относят ЭВМ типа AS/400 (Advanced Portable Model 3) - 64-разрядные «бизнескомпьютеры». Они предназначены в первую очередь для работы в финансовых структурах. В этих машинах особое внимание уделяется сохранению и безопасности данных, программной совместимости и т.д. Они могут использоваться в качестве основы серверов в локальных сетях, для управления сложными технологическими производственными процессами.

В этот период стали серийно производиться супер-ЭВМ. Первый суперкомпьютер - Cray-1 (проект С. Крея), установленный в Лос-Аламосе (США). Он выполнял до 100 млн. арифметических оп/сек. В нашей стране похожие характеристики имела супер-ЭВМ «Эльбрус-2». Многопроцессорные комплексы «Эльбрус-2» и «Эльбрус-3» имели быстродействие соответственно 100 млн. и 1 млр. Оп/сек. Нашими учеными была создана многопроцессорная система ПС-2000, в которой могло достигаться быстродействие до 200 млн. Оп/сек.

Рост степени интеграции БИС стал технологической основой повышения производительности ЭВМ . В 1985 г. начался серийный выпуск четырехпроцессорной «КРЭЙ-2» производительностью 200 млн. скалярных и 1,2 млрд. векторных оп/сек, в 1990 г. – выпуск японской четырехпроцессорной SX-344.

К числу наиболее значительных разработок конца 80-х - начала 90-х гг. относится ЭВМ «КРЭЙ-3» - 16-процессоров, 8 млрд. скалярных и 16 млрд. векторных оп/сек.

С развитием науки и техники выдвигаются новые задачи, требующие больших объемов вычислений. Особенно эффективно применение супер-ЭВМ при решении задач проектирования, в которых натурные эксперименты дорогостоящи. В этом случае ЭВМ позволяет методами моделирования получить результаты вычислительных экспериментов, обеспечивая приемлемое время и точность решения. Например, при создании супер-ЭВМ GF-11 (Gigaflop-11) с быстродействием 11 млрд. оп./сек. предварительные расчеты показали, что применение этой системы позволит решить целый комплекс новых задач. Одной из таких задач было уточнение массы протона. При использовании новой ЭВМ эта работа могла быть выполнена за 1,5-4 месяца.

Дальнейшее развитие супер-ЭВМ связывается с использованием направления массового параллелизма, при котором одновременно могут работать сотни и тысячи процессоров. В настоящее время супер-ЭВМ используются для решения крупномасштабных вычислительных задач, для обслуживания крупнейших банков данных.

Требуемое количество супер-ЭВМ для отдельной страны, такой, как Россия, должно составлять 100-200 штук, больших ЭВМ - тысячи, средних - десятки и сотни тысяч, ПЭВМ - миллионы, встраиваемых микро-ЭВМ - миллиарды.

История создания ***персональных компьютеров*** **берет свое начало с момента создания *микропроцессора (МП)* - программируемого логического устройства, выполненного на основе одной или нескольких БИС, объединяющего в себе УУ и АЛУ**. МП обладают высокой надежностью и производительностью, малыми размерами и низкой стоимостью. Это позволяет встраивать их в контрольно-измерительные и бытовые приборы, промышленно-технологическое оборудование. На его основе были созданы микро-ЭВМ. Микро-ЭВМ представляет собой вычислительную систему, включающую МП, память и устройства ввода-вывода. Микро-ЭВМ обладают достаточно высокими эксплуатационными параметрами, сравнимыми с аналогичными параметрами средних ЭВМ. Их габариты, требования к условиям эксплуатации и потребление электроэнергии находятся в границах, предъявляемых к бытовым электроприборам. Область применения микро-ЭВМ весьма широка. Они входят составной частью в измерительные комплексы, системы числового программного управления, управляющие системы различного назначения.

В 1970 г. Маршиан Эдвард Хофф (Intel) сконструировал интегральную схему, аналогичную по своим функциям центральному процессору большой ЭВМ. На одном кремниевом кристалле он сформировал минимальный по составу аппаратуры процессор, а на других БИС – оперативную и постоянную память. Так появился первый *микропроцессор* Intel-4004 (4 бита), который был выпущен в продажу в конце 1970 г. В 1973 г. фирма Intel выпустила 8-битовый микропроцессор Intel-8008, а в 1974 г. – его усовершенствованную версию Intel-8080, которая до конца 70-х годов стала стандартом для микрокомпьютеров.

Выпуск первого МП служит датой начала микропроцессорной революции. Она шла двумя путями: МП встраиваются в аппаратуру (связи, производственную, медицинскую, бытовую и т. д.) и служат основой построения соответствующих управляющих устройств; на базе МП создаются как микро-, так и супер-ЭВМ. В 80-е гг. создаются мощные 32-разрядные МП, содержащие схемы памяти, контроля и диагностики. С микропроцессорной революцией непосредственно связано одно из важнейших событий в истории ЭВМ – создание и широкое применение *персональных* ЭВМ (ПЭВМ).

С их распространением начинается фаза всеобщей компьютеризации, которая продолжается и сегодня. С 1980 г. по 1990 г. выпуск ПЭВМ в США вырос с 0,37 млн. до 25 млн. шт. и в настоящее время этот класс машин абсолютно доминирует в мировом парке ЭВМ.

В начале 1975 г. появился первый коммерчески распространяемый ПК «Альтаир-8800» (фирма MITS, США), построенный на основе МП Intel-8080. Хотя возможности его были весьма ограничены (ОП - 256 байт, клавиатура и экран отсутствовали), его появление было встречено с энтузиазмом. Покупатели этого ПК снабжали его дополнительными устройствами: монитором, клавиатурой, блоками расширения памяти и т.д. В конце 1975 г. Пол Аллен и Билл Гейтс (будущие основатели фирмы Microsoft) создали для компьютера «Альтаир» интерпретатор языка Basic, что позволило пользователям просто общаться с компьютером и легко писать для него программы. Позднее потребности в ПЭВМ возрастали невиданными темпами. Так, в 1983 г. в мире было продано 10 млн. ПК - небывалое число для вычислительной техники.

В дальнейшем ПК стали продаваться в полной комплектации, с клавиатурой и монитором. Росту объема продаж ПК способствовали также программы, разработанные для деловых применений (например, редактор текстов WordStar и табличный процессор VisiCalc, соответственно 1978 и 1979 г.г.). Эти программы сделали для делового мира покупку ПК выгодным вложением денег: с их помощью стало возможно эффективнее выполнять бухгалтерские расчеты, составлять документы и т.д. В результате оказалось, что для многих организаций необходимые им расчеты стало возможно выполнять не на больших или мини-ЭВМ, а на ПК, что значительно дешевле.

Революция в индустрии ПК была совершена фирмами IBM и Apple Computer. **При этом первым всенародным ПК стал компьютер фирмы Apple Computer.** История этой фирмы началась в 1976 году, когда Стивен Джобс и Стивен Возник создали свою первую модель, назвав ее «Apple» (яблоко). Развитием работ в данном направлении стал компьютер Apple II, созданный на 8-разрядном МП фирмы Motorola.

Apple II обладал модульной конструкцией с возможностью расширения системы. Техническая информация о нем была опубликована, что дало возможность др. производителям выпускать аппаратуру, которой пользователи могли дополнять свои компьютеры. Другими словами, архитектура Apple II была открытой, что явилось новинкой для того времени.

В 1981 году на рынке ПК появилась фирма IBM, которая во много раз превосходила по своим возможностям фирму Apple. В качестве основного МП компьютера IBM был выбран Intel-8088. В данном компьютере были использованы и другие комплектующие различных фирм, а его ПО было поручено разработать небольшой фирме Microsoft.

В 1981 г. новый ПК под названием IBM PC был представлен публике и приобрел большую популярность. Через один-два года IBM PC занял ведущее место на рынке, став фактически стандартом ПК. Сейчас такие компьютеры (совместимые с IBM PC) составляют более 90% всех производимых в мире ПК.

Основой популярности IBM PC является заложенная в нем возможность усовершенствования его отдельных частей и использования новых устройств. Фирма IBM сделала его не единым неразъемным устройством, а обеспечила возможность его сборки из частей, изготовленных независимо различными фирмами.

В 1983 г. был выпущен IBM PC XT (Personal Computer Extended Technology), имеющий встроенный жесткий диск, в 1985 г.— IBM PC AT (Personal Computer Advanced Technology) на основе нового МП Intel-80286, работающий в 3-4 раза быстрее IBM PC XT.

Однако очень скоро другие фирмы начали собирать компьютеры, совместимые с IBM PC, и продавать их дешевле аналогичных ПК IBM. Уже в 1982 г. на рынке появились точные копии компьютеров фирмы IBM, так называемые *клоны,* выпускаемые др. фирмами, поэтому сегодня мы говорим об IBM-совместимых ПК.

Принцип открытой архитектуры лишил фирму IBM монополии на выпуск ПК класса IBM PC. К 1984 г. IBM PC-совместимые компьютеры производили уже около 50 компаний, а к концу 1986 г. ежегодная продажа IBM PC-совместимых компьютеров превзошла по объему продажу оригинальных ПК IBM. При этом первые ПК на основе 32-разрядного МП Intel-80386 были выпущены уже не IBM, а фирмой Compaq Computer.

В конце 80-х г.г. IBM опять сделала попытку вернуть себе монополию на производство ПК, выпустив новую модель PS/2, отличную от IBM PC и имеющую закрытую архитектуру. Однако модель не имела успеха и в настоящее время большинство компьютеров типа IBM PC изготавливается в Юго-Восточной Азии (Тайвань, Сингапур, Южная Корея и т.д.). Впрочем, некоторые наиболее престижные ПК собираются в США и Европе, хотя многие компоненты для них все равно завозятся из Юго-Восточной Азии.

Наибольшее влияние на развитие компьютеров типа IBM PC теперь оказывает не IBM, а фирма Intel – производитель МП, и фирма Microsoft – разработчик ОС Windows и многих других используемых на IBM PC программ.

Что же касается Apple Computer, то она сохранила особенности своей модели, отличной от IBM PC. Компьютер, выполненный фирмой на самом новом в то время микропроцессоре Motorola 68000, получил имя **Macintosh** (название одного из популярных сортов американских яблок «макинтош»). Компьютер имел высококачественный графический дисплей, «мышь» и был прост в освоении даже для неподготовленного пользователя, который мог практически ничего не вводить с клавиатуры, а лишь использовать маленькие картинки-пиктограммы, обозначающие разнообразные действия. Кроме этого, впервые компьютер дополнили генератором звука и микрофоном.

В начале 1984 года Macintosh поступил в продажу. В комплекте с новейшим лазерным принтером Macintosh с его графическими возможностями стал идеальной машиной для развивающихся в то время настольных издательств. Полюбился компьютер и преподавателям школ, колледжей, университетов. А вот в деловом мире по-прежнему популярностью пользовались IBM-совместимые компьютеры.

В нашей стране в 1988 г. был начат массовый выпуск школьных персональных компьютеров и классов учебной вычислительной техники (КУВТ) «Корвет», «Электроника УКНЦ» и др., профессиональных компьютеров ДВК-3М, ДВК-4, Искра-1030, Нейрон, ЕС-1841 и др.

Выпуск МП серии Power PC, разработанных совместно фирмами Apple, IBM и Motorola, начался в 1993 г., а в 1994 г. стали появляться в продаже компьютеры фирм Apple и IBM на его базе. Большой интерес для пользователей представляет ПК Power Macintosh, который появился в 1994 г. Этот компьютер использует МП Power PC. Особенностью Power Macintosh является возможность работать с программами, написанными для IBM-совместимых персональных компьютеров.

В настоящее время на компьютерном рынке рядом с фирмами IBM и Apple Computer трудятся такие крупные фирмы, как Compaq, Packard Bell, Dell, Hewlett-Packard и др.

Причины успеха ПК. В наст. время индустрия производства компьютеров и ПО для них является одной из наиболее важных сфер экономики развитых стран. Только в США объем продаж компьютеров, услуг и ПО составляет десятки миллиардов долларов и продолжает расти. Основная причина этого - невысокая стоимость ПК и их выгодность для многих деловых применений по сравнению с большими и мини-ЭВМ. **Но имеются и другие причины:**

- простота использования, обеспеченная с помощью диалогового способа взаимодействия с компьютером, удобных и понятных пользовательских интерфейсов;
- возможность взаимодействия с компьютером без посредников и ограничений;
- относительно высокие возможности по переработке информации;
- высокая надежность и простота ремонта, основанные на интеграции компонентов компьютера;
- возможность расширения и адаптации к особенностям применения - один и тот же компьютер может быть оснащен различными периферийными устройствами и программными средствами;
- наличие ПО, охватывающего практически все сферы человеческой деятельности, а также мощных систем для разработки нового ПО.

Ограниченность **области применения персональных компьютеров:**

- при *обработке больших объемов информации* часто целесообразно совместное использование компьютеров разного уровня, где на каждом уровне решаются задачи, соответствующие его возможностям. Например, в крупном коммерческом банке обработка информации о клиентах и расчетах скорее всего потребует большую ЭВМ, а ввод данных и анализ результатов может осуществляться и на ПК;
- во многих задачах оказывается недостаточной *вычислительная мощность* ПК. Например, расчет механической прочности конструкции из нескольких сотен элементов можно сделать и на ПК, но если надо рассчитать прочность конструкции из сотен тысяч элементов, то потребуется уже большая или супер-ЭВМ;
- при *компьютерном производстве видеофильмов* ПК можно использовать для создания движущихся картинок на экране. Но для создания реалистичных фильмов и специальных видеоэффектов используются специализированные компьютеры, предназначенные для эффективной обработки трехмерных изображений и анимации.

Пятое поколение

С конца 80-х годов XX века в истории развития ЭВМ наступила пора *пятого поколения* машин. Проект машин пятого поколения разработан ведущими японскими фирмами и научными организациями, согласно этому проекту ЭВМ и вычислительные системы коренным образом отличаются от машин предшествующих поколений.

Прежде всего, их структура отличается от той, которую предложил Джон фон Нейман, и содержит:

- блок общения, обеспечивающий интерфейс между пользователем и ЭВМ на языке, близком к естественному;
- базу знаний, хранящую все необходимые для решения задач сведения о той предметной области, к которой эти задачи относятся;
- решатель, который организует подготовку программы решения задачи на основании знаний из базы знаний и исходных данных, полученных из блока общения.

ЭВМ пятого поколения должны самоорганизовываться в процессе решения задач, иметь собственную внутреннюю модель мира и активно взаимодействовать с внешней средой, распознавать образы, делать выводы из информации, уметь оперировать в нечетких ситуациях, пополнять имеющиеся знания (т.е. иметь способность обучаться), вести диалог с человеком на естественном речевом или графическом языке, иметь способность понимать содержимое базы знаний, и использовать эти знания при решении задач.

Таким образом, в этих машинах широко используются модели и средства, разработанные в искусственном интеллекте. В них, в частности, широко используются языки, характерные для представления знаний, модели знаний в виде семантических сетей, фреймов и продукций.

Разработка машин пятого поколения ведется на основе сверхбольших интегральных схем (СБИС), а также на основе перехода к супермикроэлектронике, где расстояния между элементами схем будут меньше микрона, и на основе использования достижений интегральной микрооптоэлектроники, в которой каналами связи являются световые лучи, а для преобразования электрических сигналов в световые и наоборот используются лазеры, свето- и фотодиоды. Ведутся разработки по замене кремния и биологическими системами памяти и датчиками. Благодаря достижениям современной техники только в компьютерах пятого поколения стало возможным технически реализовать идеи, высказанные еще в 70-е годы.

Отличительные черты ЭВМ пятого поколения:

- новая технология производства, возможно, не на кремнии, а на базе новых материалов;
- отказ от традиционных языков высокого уровня (Фортран, Паскаль и др.) в пользу языков с повышенными возможностями манипулирования символами и с элементами логического программирования (Пролог, ЛИСП);
- акцент на новые архитектуры (например, на архитектуру потока данных) и отход от структур фон Неймана;
- новые способы ввода-вывода, удобные для пользователя (например, распознавание речи и образов, синтез речи, обработка сообщений на естественном языке);
- искусственный интеллект, тесно связанный с исследованиями в области экспертных систем.

Компьютер фон Неймана

Потоки информации в ЭВМ.

В отчете «Предварительное обсуждение логической конструкции электронной вычислительной машины», подготовленном в 1946 г. фон Нейманом изложены основные принципы логической структуры ЭВМ (компьютера фон Неймана):

- ***принцип использования двоичной системы счисления для представления информации в ЭВМ.*** Машина должна работать не в десятичной системе счисления (как механические арифмометры), а в двоичной (бинарной). Это означает, что программа и данные должны быть записаны в коде двоичной системы, где каждое число или символ представляется определенной комбинацией нулей и единиц.
- ***программное управление работой ЭВМ.*** Программы состоят из отдельных шагов - команд; команда осуществляет единичный акт преобразования информации; последовательность команд, необходимая для реализации алгоритма, является программой; все разновидности команд, использующиеся в конкретной ЭВМ, в совокупности являются языком машины или системой команд машины.
- ***принцип условного перехода.*** Возможность перехода в процессе вычислений на тот или иной участок программы в зависимости от промежуточных, получаемых в ходе вычислений результатов; реализация принципа условного перехода позволяет осуществлять в программе циклы с автоматическим выходом из них, итерационные процессы и т.п. Благодаря принципу условного перехода число команд в программе получается во много раз меньше, чем число выполненных машиной команд при исполнении данной программы за счет многократного вхождения в работу участков программы.
- ***принцип хранимой программы.*** Программа, которая управляет последовательностью выполнения операций, должна храниться в памяти машины. Там же должны храниться исходные данные и промежуточные результаты.
- ***принцип иерархичности запоминающих устройств.*** Память компьютера следует организовать по иерархическому принципу, т. е. она должна состоять, по крайней мере, из двух частей: быстрой, но небольшой по емкости (оперативной) и большой (медленной) внешней.

Классическая структурная схема ЭВМ, построенной в соответствии с принципами фон Неймана, приведена на след. рис.

КЛАССИЧЕСКАЯ СТРУКТУРНАЯ СХЕМА ЭВМ

На схеме представлены: *память*(*оперативная* и *внешняя*), состоящая из перенумерованных ячеек (*номер ячейки* называют *адресом*); *процессор*, включающий в себя *устройство управления* (УУ) и *арифметико-логическое устройство* (АЛУ); *устройство ввода*; *устройство вывода*. Эти устройства соединены каналами связи, по которым передается информация (управляющая – прерывистые стрелки, данные – непрерывные стрелки).

Функции памяти:

- приём информации из других устройств;
- запоминание информации;
- выдача информации по запросу в другие устройства машины.

Функции процессора:

- обработка данных по заданной программе путем выполнения арифметических и логических операций;
- программное управление работой устройств компьютера.

Та часть процессора, которая выполняет команды, называется арифметико-логическим устройством, а другая его часть, выполняющая функции управления устройствами, называется устройством управления. Обычно эти два устройства выделяются чисто условно, конструктивно они не разделены. АЛУ реализует важную часть процесса обработки данных, заключающуюся в выполнении набора простых операций. Операции АЛУ подразделяются на три основные категории: *арифметические*, *логические* и *операции над битами*. Арифметической операцией называют процедуру обработки данных, аргументы и результат которой являются числами (сложение, вычитание, умножение, деление,...). Логической операцией именуют процедуру, осуществляющую построение сложного высказывания (операции И, ИЛИ, НЕ,...). Операции над битами обычно подразумевают сдвиги.

Алгоритмом решения задачи численным методом называют последовательность арифметических и логических операций, которые надо произвести над исходными данными и промежуточными результатами для получения решения задачи. Алгоритм можно задать указанием, какие следует произвести операции, в каком порядке и над какими данными. **Описание алгоритма в форме, воспринимаемой ЭВМ, называется *программой*. Программа состоит из команд.**

Каждая команда предписывает определенное действие и указывает, над какими словами (операндами) это действие производится. Программа представляет собой совокупность команд, записанных в определенной последовательности, обеспечивающей решение задачи на ЭВМ. Для того чтобы УУ могло воспринимать команды, они должны быть закодированы в цифровой форме.

Автоматическое управление процессом решения задачи достигается на основе принципа программного управления, который составляет главную особенность ЭВМ.

Согласно принципу хранимой программы программа хранится в памяти наравне с числами. В команде указываются не участвующие в операциях числа, а адреса ячеек ОП, в которых они находятся и куда помещается результат операции. Команды выполняются в последовательном порядке, кроме команд перехода, изменяющих этот порядок безусловно или при выполнении некоторого условия, задаваемого в виде равенства нулю, положительного или отрицательного результата предыдущей команды или отношения типа <, =, > для указываемых командой чисел. Благодаря наличию команд условного перехода ЭВМ может автоматически изменять ход выполняемого процесса, решать сложные логические задачи.

При помощи устройства ввода программа и исходные данные считываются и переносятся в ОП.

Как правило, процесс выполнения команды разбивается на следующие этапы:

- из ячейки памяти, адрес которой хранится в счетчике команд УУ, выбирается очередная команда, содержимое счетчика команд при этом увеличивается на длину команды;
- выбранная команда передается в УУ на регистр команд;
- УУ расшифровывает адресное поле команды;
- по сигналам УУ операнды считываются из памяти и записываются в АЛУ на регистры операндов;
- УУ расшифровывает код операции и выдает в АЛУ сигнал выполнения соответствующей операции над данными;
- результат операции либо остается в процессоре, либо отправляется в память, если в команде был указан адрес результата;
- все предыдущие этапы повторяются до достижения команды «стоп».

Данные, типы элементарных данных.

Непрерывная и дискретная информация.

Информация о различных природных явлениях и технологических процессах воспринимается человеком в виде тех или иных полей. Математически такие поля представляются с помощью функций $y = f(x,t)$, где t – время, x – точка, в которой измеряется поле, y – величина поля в этой точке. При измерениях поля в фиксированной точке функция вырождается в функцию времени $y(t) = f(a,t)$, которую можно изобразить в виде графика.

В большинстве случаев скалярные величины, входящие в соотношение, могут принимать непрерывный ряд значений, измеряемых вещественными числами. Под непрерывностью здесь понимается то, что рассматриваемые величины могут изменяться сколь угодно мелкими шагами. Ввиду этого **представляемую таким способом информацию называют *непрерывной* (*аналоговой*) информацией.**

Если применительно к той же самой информации о поле $y = f(x,t)$ установить минимальные шаги изменения всех характеризующих ее скалярных величин, то получим ***дискретное* представление информации (*дискретная* информация).** Поскольку точность измерений всегда ограничена, то, даже имея дело с непрерывной информацией, человек воспринимает ее в дискретном виде. Но любая непрерывная информация может быть аппроксимирована дискретной информацией с любой степенью точности, поэтому говорят об *универсальности* дискретной формы представления информации.

Результаты измерения скалярных величин представляются в итоге в числовом виде, а поскольку при заданной точности измерений эти числа представимы в виде конечных наборов цифр (с запятой или без нее), то **дискретную форму представления информации отождествляют с *цифровой информацией.***

Абстрактные алфавиты.

Цифровая информация представляет собой частный случай так называемого *алфавитного способа* представления дискретной информации. Его основой является произвольный фиксированный конечный набор символов любой природы, называемый *абстрактным алфавитом* или просто *алфавитом.*

Совокупность десятичных цифр вместе с запятой для отделения дробной части числа можно рассматривать как частный случай абстрактного алфавита с 11 символами – *буквами* этого алфавита. Другой пример – алфавит естественного человеческого языка, языка математических и других научных текстов, др.

Кодирование.

При обработке информации часто необходимо представлять средствами одного алфавита буквы других алфавитов. Такое представление носит в информатике наименование *кодирования.* Проблема решается просто, если требуется закодировать буквы алфавита *X* с меньшим числом букв, чем у кодирующего алфавита *Y.* Если, например, *X* — алфавит десятичных цифр, а *Y* — обычный русский алфавит, то для кодирования *X* в *Y* достаточно положить 0 = а, 1=б, 2 = в, 3 = г, ..., 9 = к. Конечно, возможны и другие способы кодирования, одним из наиболее естественных из которых является просто замена десятичных цифр их русскими названиями: нуль, один, два и т. д.

При кодировании алфавитов с большим числом букв в алфавитах с меньшим числом букв использование для кодирования последовательностей букв является обязательным условием для возможности различения кодов различных букв. Так, буквы русского алфавита можно закодировать парами десятичных цифр, например, а = 01, б = 02, ..., к = 10, л = 11, ...

Двоичный алфавит.

Простейшим абстрактным алфавитом, достаточным для кодирования любого другого алфавита, является алфавит, состоящий из двух букв. Такой алфавит носит наименование *двоичного,* а две его буквы чаще всего принято обозначать цифрами *0* и *1*. Величина, способная принимать лишь два различных значения, представляет собой ***информационный атом,*** получивший наименование ***бит.***

Байтовый алфавит.

Ввиду своей простоты двоичный алфавит наиболее широко распространен в технических устройствах, в первую очередь в ЭВМ. Для кодирования же алфавитов, которыми привык пользоваться человек, употребляются последовательности двоичных цифр. Легко видеть, что последовательностями из n двоичных цифр можно закодировать $2n$ различных символов. При $n = 8$ их число равно 256, что оказывается достаточным для кодирования большинства встречающихся на практике алфавитов.

Последовательность из 8 двоичных цифр получила в связи с этим наименование *байта*. Составляемый же подобными последовательностями алфавит из 256 букв называют *байтовым алфавитом.* В практике использования ЭВМ в международном масштабе укоренился единый стандарт байтового кодирования строчных и прописных букв латинского алфавита, знаков препинания, десятичных цифр с десятичной запятой (точкой), а также ряда математических символов (знаки арифметических и логических операций, знаки равенства и неравенства и др.). Специальный код закрепляется за знаком *пробела.*

Мощность **(число букв)** байтового алфавита достаточна для представления, кроме этих символов, строчных и прописных букв русского алфавита, отличных по написанию от латинских букв. Остается резерв и для кодирования других символов, например греческих букв.

Слова и абстрактные языки.

Любая конечная последовательность букв в алфавите *X* называется *словом.*

Для того, чтобы из всего получаемого таким образом множества слов выделить правильные в каком-то смысле слова, над исходным алфавитом *X* строится так называемый *формальный язык.*

Кроме алфавита *X* формальный язык задается своей формальной *грамматикой,* которая представляет собой конечную совокупность формальных правил, с помощью которых порождаются все слова данного языка и только такие слова.

Данные, элементарные данные.

Информация, с которой имеют дело ЭВМ, называется ***данными,*** которые разбиваются на отдельные составляющие, называемые ***элементарными данными*** или ***элементами данных.*** Употребляются элементы данных различных типов. ***Тип данных*** (элементарных) зависит от значений, которые эти данные могут принимать. В современной информатике среди различных типов элементарных данных наиболее употребительными являются ***целые и вещественные числа, слова и булевы величины***.

Прежде всего **различают двоичное и двоично-десятичное представления чисел**. В двоичном представлении используется *двоичная система счисления* с фиксированным числом двоичных разрядов (чаще всего 32 разряда, включая разряд для представления знака числа). Если нулем обозначать плюс, а единицей – минус, то 00001010 означает целое число $+(2+2^3)=+10$, а 10001100 число $-(2^2+2^3)=-12$ (для простоты взято 8-разрядное представление).

В случае вещественных чисел употребляются две формы представления: с *фиксированной* и с *плавающей запятой.* В первом случае просто заранее уславливаются о месте нахождения запятой, не указывая ее в коде числа. Например, если условиться, что запятая стоит между 3-м и 4-м разрядами справа, то код 00001010 будет означать число .

$$00001{,}010_2 = (1+0\cdot 2^{-1}+0\cdot 2^{-2}+0\cdot 2^{-3}) = 1{,}25^{10}\cdot 1+\frac{1}{2^2}=1{,}25$$

Во втором случае код числа разбивается на два кода в соответствии с представлением числа в виде $x = a\cdot 2^b$. При этом число *a* (со знаком) называется ***мантиссой***, а число *b* (со знаком) – *характеристикой* числа *x.* О положении кода характеристики и мантиссы (вместе с их знаками) в общем коде числа также условливаются заранее. Для экономии числа разрядов в характеристике *b* ее часто представляют в виде $b = 2^k b_1$ где *k* – фиксированная константа (обычно *k* = 2). Вводя еще одну константу *m* и полагая $b = 2^k b_2 - m$, можно избежать также использования в коде характеристики знака (при малых число *b* отрицательно, а при больших – положительно).

В двоично-десятичном представлении обычные десятичные цифры (а также запятая и знак) кодируются двоичными цифрами. При этом часто используется так называемый *упакованный код,* когда с помощью одного байта кодируется не одна, а две десятичные цифры. Подобное представление позволяет кодировать числа любой значности.

Тип данных «произвольное слово во входном алфавите» не нуждается в специальных пояснениях. Единственное условие – необходимость различать границы отдельных слов. Это достигается использованием специальных ограничителей и указателей длины слов.

Тип булев присваивается элементарным данным, способным принимать лишь два значения: «истина» и «ложь». Для представления булевых величин обычно используется двоичный алфавит.

Переменные и постоянные величины (данные).

Для идентификации данных употребляются специальные наименования, называемые *идентификаторами.* Они являются аналогами переменных величин, фигурирующих в математике. Область значений, которые может принимать эта переменная, определяется типом элемента данных, именем которого она является. **В качестве идентификаторов элементарных данных принято использовать любые конечные последовательности латинских букв и десятичных цифр, начинающиеся буквой.**

В отличие от переменных величин, для которых требуется различать их *имя* и *значение,* для постоянных величин ее значение может служить и ее именем. Однако часто полезно именовать постоянные величины идентификаторами (например, число $e \approx 2{,}718$ и др.), имея в виду, что такого рода идентификатор представляет собой *константу* и может принимать лишь единственное значение.

Структуры данных и структурированная память.

На практике в большинстве случаев данные образуют ту или иную *структуру. Структурирование данных* задается прежде всего с помощью различного рода *отношений порядка (упорядоченности).* Простейший вид упорядоченности - нумерация данных с помощью последовательных целых чисел. Идентификаторы упорядоченных таким образом данных образуются обычно путем приписывания какому-либо фиксированному идентификатору специального *идентификатора индекса,* значения которого пробегают заданный набор целых чисел.

Возникающий таким образом идентификатор вида x_i или $x[i]$ где *i* пробегает целые числа от *m* до *n*, идентифицирует упорядоченный набор данных, называемый обычно *одномерным массивом.* Двухиндексный идентификатор $x_{i,j}$ вида $x[i,j]$ или идентифицирует *двумерный массив,* трехиндексный – *трехмерный массив* и т. д. В упорядоченных таким образом массивах возникает естественное *отношение следования:* так, следующим по индексу *j* для элемента $x_{i,j}$ будет элемент $x_{i,j+1}$, а предыдущим – элемент $x_{i,j-1}$. Если индекс *j* пробегает значения от *p* до *q,* то для *p* не существует предыдущего, а для *q* – следующего значения индекса.

Если границы изменений индексов задаются константами, то говорят, что соответствующий массив – *прямоугольный,* поскольку в двумерной целочисленной решетке область возможных значений индексов в этом случае представляет собой прямоугольник (термин *прямоугольный* применяется также для массивов любой размерности). Области значений индексов могут задаваться и более сложными способами.

Например, соотношения $p \le i \le j \le q$ задают *треугольный массив.* Массивы обычно предполагаются *однородными* , состоящими из элементов одного типа. Одномерные однородные массивы называют *векторами,* двумерные - *матрицами,* а составляющие их элементарные данные - их *компонентами* (*элементами*). В общем случае индексы могут пробегать не обязательно последовательно все целые числа с шагом, равным единице, а любые их конечные подмножества, в том числе с переменной величиной шага.

Помимо однородных массивов на практике часто приходится иметь дело с данными различных типов. Рассмотрим, например, содержание анкеты, применяемой при учете кадров. Одни графы этой анкеты заполняются **словами**, другие – **числами**, третьи – **булевыми величинами (*да, нет*).** В информатике каждая совокупность данных, характеризующая тот или иной объект или явление, называется *записью.* Кадровая анкета представляет собой один из возможных видов подобной записи. Совокупность записей (обычно одного и того же типа) носит наименование *файла.*

Если запись характеризует тот или иной объект, то отдельные ее позиции представляют собой различные свойства этого объекта, или его *атрибуты.* Если запись состоит из *n* атрибутов, то каждый файл записей такого типа задает *n-местный предикат (n-местное отношение)* $f(x_1, x_2, ..., x_n)$ на множестве всевозможных значений этих атрибутов. Этот предикат считается истинным для значений атрибутов $x_1 = a_1,\ x_2 = a_2,\ ...,\ x_n = a_n$, если в заданном файле существует запись $\langle a_1, a_2, ..., a_n \rangle$ и ложным в противном случае. Таким образом, всякий набор файлов задает некоторую ***модель данных***.

Записи – это подмножества множества данных длины *n*, имеющие специальный вид. В общем случае структура данных может быть задана в виде произвольной системы *N* подмножеств множества *M* (упорядоченных или нет). Подмножества системы *N* получают при этом некоторые имена и включаются в рассмотрение в качестве составных данных наряду с исходными (элементарными) данными множества *M*.

Например, любая книга без рисунков может рассматриваться как упорядоченная последовательность букв (в число которых включаются знаки препинания и другие специальные символы, включая символ пробела). Эти буквы объединяются в слова, которые можно рассматривать как составные объекты первого уровня, слова – в предложения, предложения – в параграфы, а параграфы – в главы.

В структурированной т.о. книге объект (данное) каждого уровня (за исключением самого верхнего) входит в один и только один объект следующего (более высокого) уровня. **Данные с такой структуризацией принято называть *деревьями* или *иерархическими структурами*. Составной объект самого верхнего уровня называется *корнем*, а объекты самого нижнего уровня – *листьями* этого дерева.**

Виды памяти и ее характеристики.

Запоминающие устройства (ЗУ), или память, хранят команды и данные. Память делится на *ячейки*, или *слова*, состоящие обычно из фиксированного числа двоичных разрядов-битов (последовательность из 8 двоичных цифр – *байт*). Ячейки нумеруются последовательными целыми числами n=0,1,2,… Номера ячеек - *адреса*. Объем ЗУ выражается в *килобайтах* (Кбайт или сокращенно К), *мегабайтах* (Мбайт), *гигабайтах* (Гбайт), и т.д. Поскольку адресация большинства ЗУ осуществляется в двоичной системе счисления, в качестве килобайта удобно брать 1024 = 2**10 байт. Соответственно, под мегабайтом понимается величина 2**20 байт.

ЗУ с *произвольным доступом* характеризуются одним и тем же *временем доступа* в произвольную ячейку независимо как от ее адреса, так и от предыдущих обращений в память. Под временем доступа здесь понимается время, необходимое для установления соединения данной ячейки с некоторым буферным регистром. Добавляя к этому времени время, необходимое для фактического перемещения информации из ЗУ в регистр или обратно, получают соответственно времена *цикла чтения* и *цикла записи.* Возможность в любой момент времени считать или записать информацию в любую ячейку памяти делает ЗУ с произвольным доступом естественным партнером центрального процессора в качестве оперативного хранилища информации. Это привело к закреплению за ними наименования оперативного запоминающего устройства (ОЗУ).

При высоком быстродействии процессора время цикла для ОЗУ может явиться ограничивающим фактором для скорости работы системы ОЗУ-процессор. В качестве выхода из этого положения строятся двухуровневые памяти, когда относительно медленное ОЗУ большого объема взаимодействует непосредственно не с процессором, а со специальной сверхоперативной памятью (СОЗУ), представляющей собой память с произвольным доступом относительно небольшого объема, но существенно более быструю, чем ОЗУ.

В ЗУ *последовательного доступа* доступ осуществляется последовательно в порядке роста или убывания адресов ячеек. Поэтому после обращения в ячейку с адресом *n* следующее соединение будет установлено либо с (*n*+1)-й, либо с (*n*–1)-й ячейкой. Последовательные ЗУ характеризуются помимо объема также скоростью передачи информации, которая в этом случае существенным образом зависит как от ее месторасположения в памяти, так и от адреса предыдущего соединения с ЗУ. Пример - магнитная лента.

ЗУ с *прямым доступом* комбинируют черты устройств с произвольным и последовательным доступом. Обладая возможностью перехода от одного адреса к любому другому, они вместе с тем гораздо быстрее открывают ячейки с последовательными адресами. Поэтому в их характеристику, помимо максимального времени произвольного доступа, входит, как и в последовательных ЗУ, независимая от этого времени скорость передачи информации.

При построении ЭВМ часто оказывается удобным использовать **ЗУ *магазинного типа*, называемые также *стеками:*** при вводе (записи) нового слова уже заполненные слова сдвигаются на один шаг в глубь магазина; при выводе (чтении) процесс протекает в обратном направлении. Т.о., реализуется принцип: последнее слово из записанных читается первым.

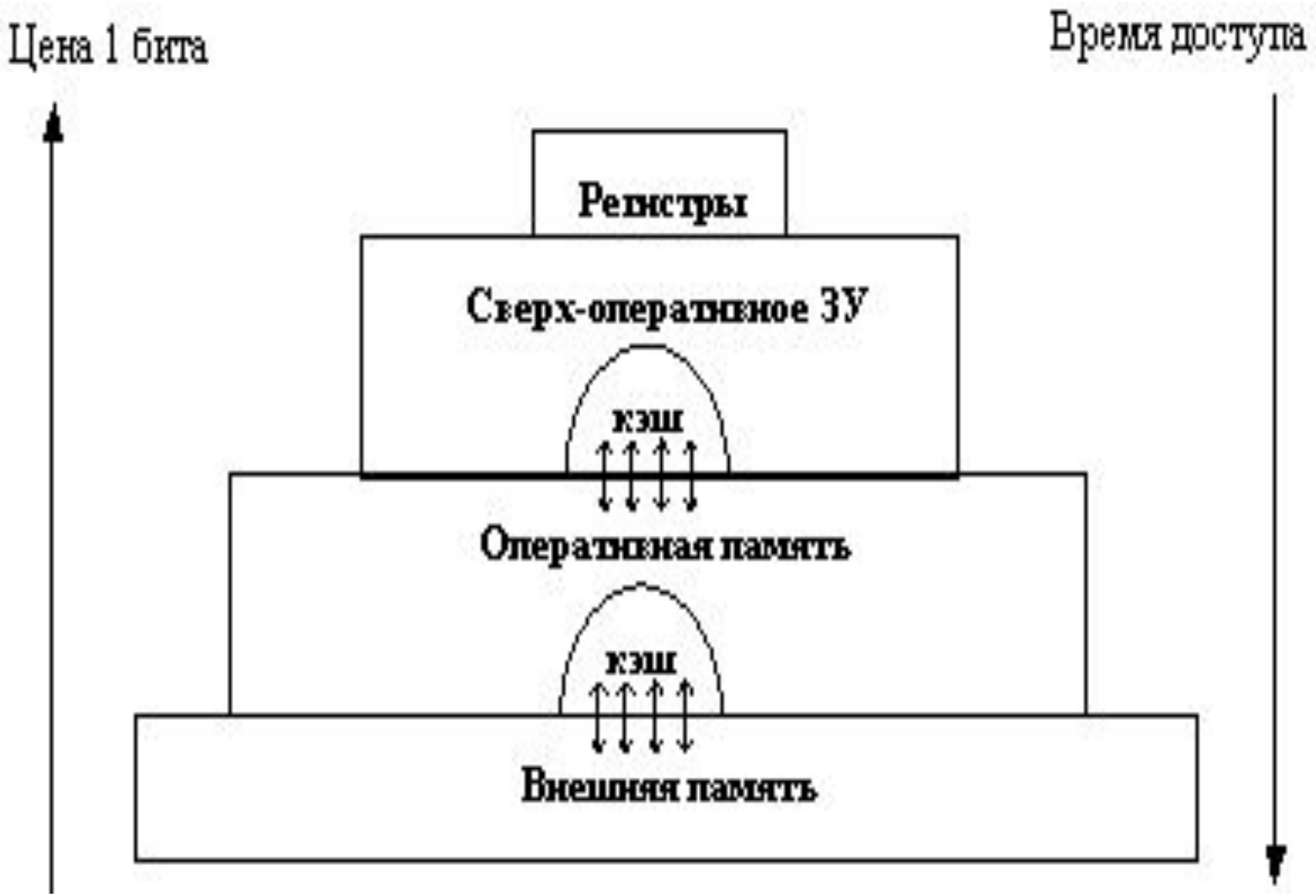
Для ряда приложений очень удобна так называемая ***ассоциативная память.*** Запись в нее осуществляется в произвольную ячейку, выбираемую самим ЗУ. Доступ к нужному элементу данных осуществляется не по адресу, а по запоминаемому вместе с этим элементом *признаку.*

Иерархия запоминающих устройств. Принцип кэширования данных. Память ЭВМ представляет собой иерархию запоминающих устройств (внутренние регистры процессора, различные типы сверхоперативной и оперативной памяти, диски, ленты), отличающихся средним временем доступа и стоимостью хранения данных в расчете на один бит (след. слайд). Пользователю хотелось бы иметь и недорогую и быструю память. Кэш-память представляет некоторое компромиссное решение этой проблемы.

Кэш-память **-** это способ организации совместного функционирования двух типов ЗУ, отличающихся временем доступа и стоимостью хранения данных, который позволяет уменьшить среднее время доступа к данным за счет динамического копирования в «быстрое» ЗУ наиболее часто используемой информации из «медленного» ЗУ.

Кэш-памятью часто называют не только способ организации работы двух типов ЗУ, но и одно из устройств – «быстрое» ЗУ. Оно стоит дороже и, как правило, имеет сравнительно небольшой объем. Важно, что механизм кэш-памяти является прозрачным для пользователя, который не должен сообщать никакой информации об интенсивности использования данных и не должен никак участвовать в перемещении данных из ЗУ одного типа в ЗУ другого типа, все это делается автоматически системными средствами.

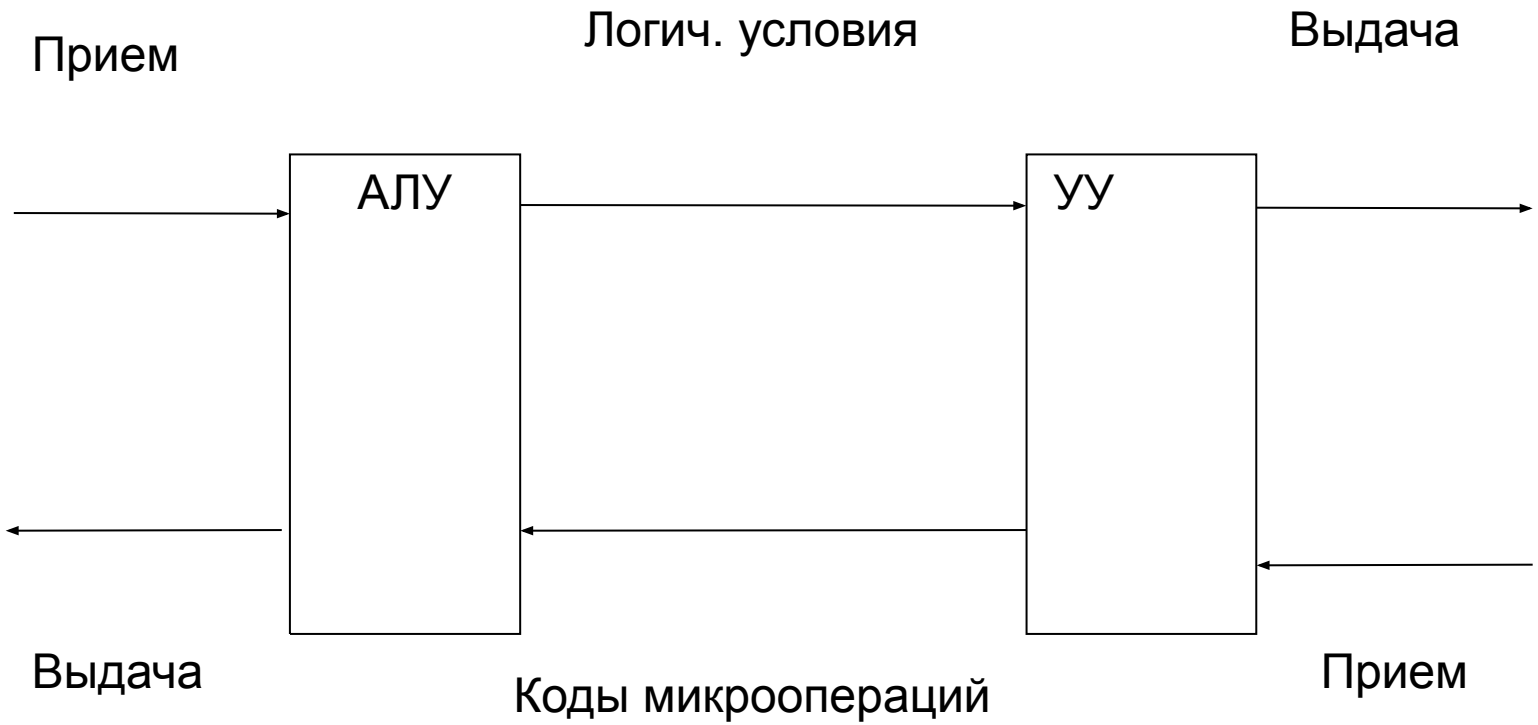
Иерархия ЗУ


Цена 1 бита
Время доступа
Регистры
Сверх-оперативное ЗУ
кэш
Оперативная память
кэш
Внешняя память

Процессор

Мультипроцессорность

Схема процессора



Процессором называется устройство, способное выполнять некоторый заданный набор операций над данными в структурированной памяти и вырабатывать значение заданного набора логических условий над этими данными. Стандартная схема процессора состоит из двух устройств, называемых *арифметико-логическим устройством* и *устройством управления* .

В схему АЛУ включается структурированная память, состоящая из регистров, к которым может добавляться один или несколько стеков. С помощью специальных комбинационных схем в структурированной памяти может осуществляться тот или иной набор преобразований.

Операции, выполняемые АЛУ за один такт синхронизирующего генератора - *микрооперации,* соответствующий их выполнению такт – *микротакт.* Выбор той или иной микрооперации осуществляется путем подачи ее кода на специальный управляющий вход АЛУ. Как правило, в состав микроопераций АЛУ включается *очистка* регистров и стеков, *пересылки данных* между регистрами и стеками, *сдвиги* (вправо или влево) двоичного кода на регистрах, а также *накапливающее суммирование.* Для этой операции выделяются специальные регистры, называемые *накапливающими сумматорами* (аккумуляторами). При передаче кода некоторого числа *x* в аккумулятор, содержавший число *y,* происходит суммирование (с учетом знаков) этих чисел, а сумма *x* + *y* замещает в аккумуляторе его первоначальное содержимое *y.*

Помимо описанных *внутренних микроопераций,* в АЛУ реализуются также *внешние микрооперации,* осуществляющие *прием* и *выдачу* данных, т. е. *обмен данными* между АЛУ и внешним миром. Кроме того, АЛУ может формировать дополнительно выходные сигналы в виде набора *логических условий,* выражающих те или иные свойства содержимого памяти АЛУ. Примеры таких свойств: «коды в регистрах *A* и *B* одинаковы», «число в регистре *A* отрицательно» и т. д.

УУ процессора на основе входных сигналов, в качестве которых служат формируемые АЛУ наборы логических условий, формирует выходные сигналы – коды микроопераций АЛУ. Кроме того, УУ может иметь также каналы обмена с внешним миром. Каналы выдачи и приема УУ и АЛУ обычно объединяются в каналы обмена процессора с внешним миром.

Через канал приема в УУ поступают коды *команд* (инструкций) на выполнение тех или иных *операций процессора,* а через канал выдачи – запросы на очередные команды, либо сигнал *останова,* который прекращает работу процессора. Операция процессора обычно не сводится к одной микрооперации, а индуцируется некоторой их последовательностью. Такие последовательности определяются *микропрограммами,* встраиваемыми или запоминаемыми в УУ. В случае запоминаемых микропрограмм их можно оперативно менять, настраивая процессор на различные наборы операций (команд). В этом случае говорят, что процессор имеет *мягкую (перестраиваемую) архитектуру*.

Работа процессора может осуществляться двумя различными способами. В первом из них процессор сам извлекает последовательно команды из *программы,* запомненной в быстродействующей памяти – внешней по отношению к процессору. Все необходимые данные (исходные, промежуточные и выходные) также хранятся в этой памяти.

Команды, помимо кода операции, содержат адреса данных, необходимых для их исполнения. Извлекая из памяти очередную команду и анализируя ее, процессор тут же извлекает необходимые для ее выполнения данные. В этом случае принято говорить, что процессор управляется *потоком команд.* Другой способ состоит в том, что данные, поступающие в АЛУ процессора из внешнего источника, запоминаются в нем, анализируются и вызывают (с помощью УУ) выборку из быстродействующей памяти последовательности команд для их обработки. В этом режиме процессор управляется *потоком данных.* Быстродействие процессора должно быть достаточно велико, чтобы обработать поступившие данные и выдать результат обработки до поступления очередной порции данных.

Типы процессоров

Арифметические процессоры. Арифметическими называются процессоры, предназначенные в первую очередь для автоматизации вычислительных операций той или иной степени сложности.

Для сложных научных, проектно-конструкторских и планово-экономических расчетов наиболее употребительны процессоры, выполняющие полный набор арифметических операций над многозначными числами в двоичной системе счисления параллельным способом.

Выбор двоичной системы и параллельного способа выполнения операций (над всеми разрядами одновременно) обусловлен тем, что при этом достигаются наибольшая скорость вычислений и наилучшее использование оборудования (прежде всего – памяти).

Магистральные процессоры. При выполнении последовательности поступающих команд процессор реализует целый ряд повторяющихся операций: он должен воспринять код очередной операции, адреса, по которым хранятся операнды, и адрес, по которому должен быть размещен результат операции, принять и разместить на регистрах операнды, выполнить операцию, разместить ее результат и подготовиться к восприятию следующей команды (выработать адрес, по которому за ней надо обратиться).

Поскольку все составные части процесса выполнения любой команды выстроены в цепочку, имеет смысл их выполнять на *конвейере*, представляющем собой ряд соединенных друг с другом *специализированных процессоров,* каждый из которых выполняет лишь свою часть работы, после чего передает обрабатываемую команду следующему процессору по цепочке. Сам же он получает от предыдущего процессора следующую команду для исполнения соответствующей ее части. Темп выполнения команд при этом существенно ускоряется.

Рассмотренный *конвейерный метод* представляет собой частный случай *магистрального метода.* В отличие от рассмотренного простого конвейера, в *магистрали* на отдельные составные части раскладывается также выполнение основной операции. Поскольку такие разложения для разных операций, вообще говоря, различны, в магистральном методе используется обычно несколько конвейерных линий с гибко перестраиваемыми связями между ними.

В идеале на магистральном процессоре последовательность команд может обрабатываться в темпе следования отдельных микроопераций или в темпе – один рабочий такт процессора (микротакт) на одну команду (при достаточной скорости внешних обменов).

Специализированные процессоры. Если ЭВМ предназначается для работы, в которой часто приходится использовать определенные виды сложных операций, в ее состав включают специализированные процессоры, предназначенные для быстрого выполнения этих операций.

Наибольшее распространение получили *векторные* и *матричные процессоры,* в состав которых включается обычно несколько десятков *элементарных процессоров* для одновременного выполнения тех или иных операций над отдельными компонентами вектора или матрицы.

Наряду с векторно-матричными процессорами достаточно широкое распространение получили процессоры, ориентированные на задачи спектрального анализа сигналов (быстрое преобразование Фурье), задачи интерполяции и на ряд других применений.

Микропроцессоры. Успехи микроэлектроники позволили создать процессоры, полностью размещаемые на одном кристалле. Вначале разрядность таких процессоров была очень низкой (4 двоичных разряда), что фактически не позволяло использовать их как отдельные устройства. По мере увеличения уровня интеграции оказалось возможным повысить разрядность однокристальных микропроцессоров до 8, а потом и до 32 двоичных разрядов.

Важнейшим качеством микропроцессоров, помимо миниатюрных размеров и малого энергопотребления, является их относительно низкая стоимость. Благодаря этому качеству микропроцессоры и строящиеся на их основе микро-ЭВМ и ПЭВМ находят широкий класс применений, для которых применение ЭВМ прежних поколений было экономически невыгодно.

Оценка производительности процессоров. Производительность процессоров определяется обычно количеством операций (команд), которые они могут выполнить за одну секунду. Однако время выполнения различных операций может довольно сильно разниться. Поэтому при определении производительности указывают *среднее быстродействие* на *смеси* различных операций, взятых в определенных пропорциях друг к другу.

Если $p_1, \ldots, p_n$ – относительные доли этих операций в смеси, а $t_1, \ldots, t_n$ – время выполнения каждой из этих операций (в сек.), то среднее быстродействие вычисляется по формуле

$$\mathbf{P} = \frac{\sum_{i=1}^{n} p_i}{\sum_{i=1}^{n} p_i t_i}$$

Мультипроцессирование. В многопроцессорных вычислительных системах центральные процессоры закрепляются и своевременно перераспределяются между различными программными модулями и различными потоками данных. Наиболее просто этот вопрос решается в том случае, когда каждый ЦП решает свою собственную независимую задачу. Если несколько ЦП используются для решения одной и той же задачи, то должно производиться *распараллеливание* вычислительного процесса.

Необходимо отметить, что распараллеливание допускают не все процессы. В тех случаях, когда оно возможно, различают два основных типа распараллеливания.

Первый тип использует для всех процессоров один и тот же поток команд, но различные потоки данных. Например, при сложении двух матриц можно разделить между процессорами строки этих матриц и выполнить параллельно сложение различных пар строк с помощью одного и того же программного цикла.

Второй тип распараллеливания использует для каждого процессора свой программный модуль, выполняемый над одними и теми же или различными данными. Поскольку разные модули исполняются, как правило, за разное время, возникает задача их *синхронизации.* Смысл ее состоит в том, что очередной программный модуль запускается в работу лишь после того, как для него подготовлены (в результате работы предшествующих модулей) все необходимые исходные данные. Подобная взаимозависимость модулей определяет одновременно и возможность их параллельного исполнения.

Мультизадачность, разделение времени.

Реальное время.

Первые ЭВМ могли выполнять только одну программу, расположенную в их памяти. В последующим широкое распространение получил режим *пакетной обработки данных*, суть которого состоит в том, что в ЭВМ вводится одновременно не одна программа, а пакет программ. Используя то обстоятельство, что в современных ЭВМ обмены с внешними устройствами могут выполняться одновременно с работой центрального процессора, ЭВМ одновременно работает с несколькими программами. Подобный режим, называемый режимом *мультипрограммирования,* обеспечивает лучшую загрузку оборудования и увеличение пропускной способности ЭВМ в целом. В процессе такой работы переключение центрального процессора с одной программы на другую осуществляется не только по их окончании, но и в том случае, если данные для решаемой задачи еще не введены, или промежуточные результаты ее решения необходимо вывести на устройство вывода или во внешнюю память. Одновременно с таким *прерыванием* решаемой задачи формируется соответствующее задание необходимому внешнему устройству, которое и выполняется параллельно с работой центрального процессора по выполнению одной из программ.

Второй режим работы компьютера – режим *разделения времени* обусловлен появлением пользовательских терминалов и позволяет одновременно обслуживать большое количество пользователей, взаимодействующих с ЭВМ в *диалоговом (интерактивном)* режиме. С этой целью каждому пользователю выделяются *элементарные порции (кванты) времени,* в течение которого ЭВМ его и обслуживает. Поскольку время реакции пользователя на ту или иную ситуацию намного превышает время реакции ЭВМ, у каждого из них создается впечатление, что он один обслуживается компьютером.

Третий режим работы компьютера – режим *реального времени* получил распространение в связи с широким использованием ЭВМ для управления внешними по отношению к компьютеру оборудованием и процессами: станками, технологическими процессами и т. п. В этом случае работа компьютера управляется внешним потоком данных, при этом производительность ЭВМ должна быть достаточной для того, чтобы обработать поступившие данные до поступления новой, очередной порции данных, а также на основе результатов обработки выдать при необходимости управляющее воздействие на внешнее оборудование за приемлемое, обусловленное особенностями работы этого оборудования, время.

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Назначение операционной системы

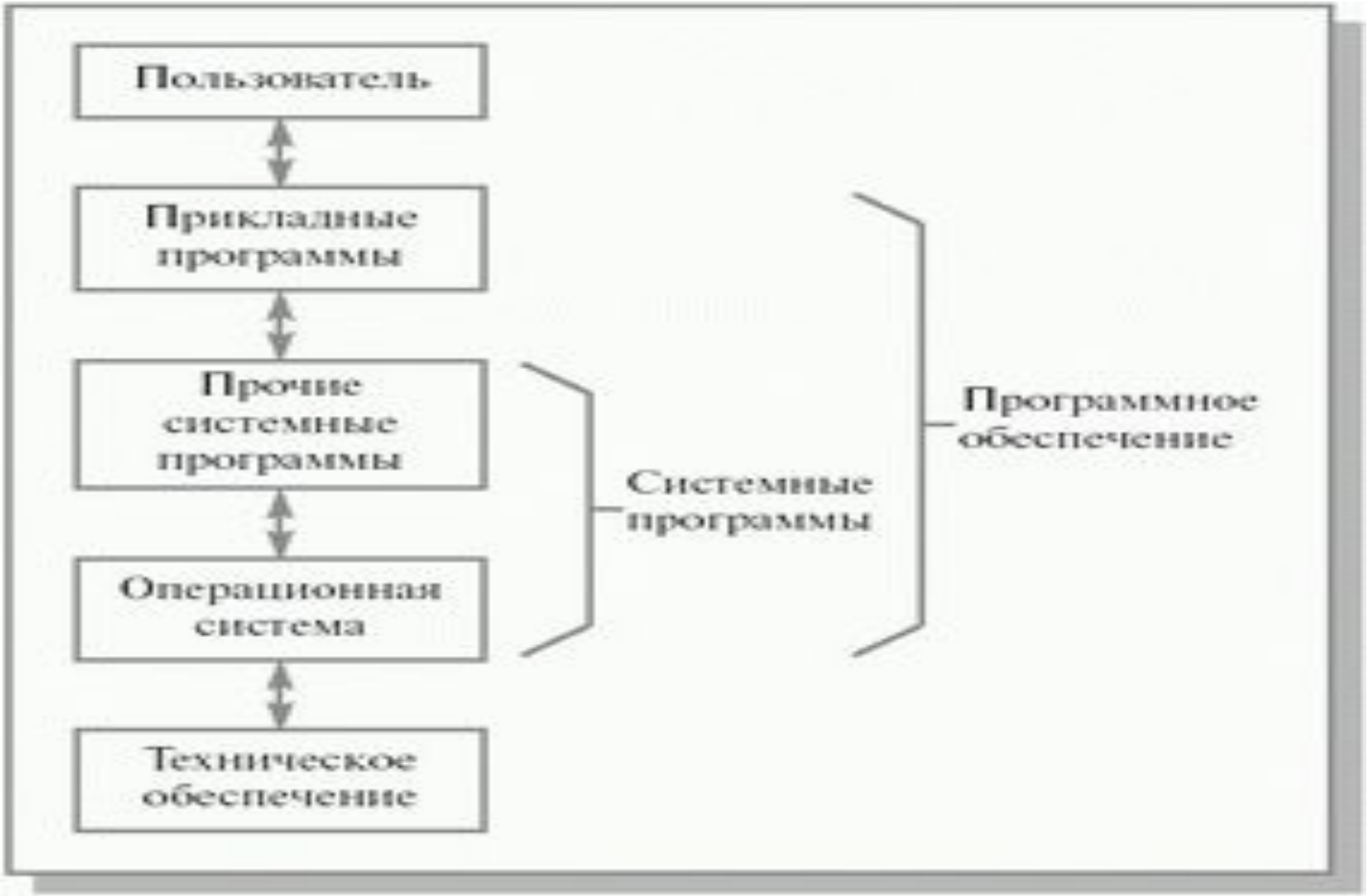
На современном уровне при рассмотрении ЭВМ выделяют их техническую часть **(HardWare)** и программное обеспечение **(SoftWare),** центральная часть которого и представлена **операционной системой (ОС).** Все программное обеспечение (ПО) принято делить на две части: прикладное и системное.

К прикладному программному обеспечению, как правило, относят разнообразные бизнес-программы, игры, текстовые процессоры и т. п.

Под системным программным обеспечением понимают программы, способствующие функционированию и разработке прикладных программ.

На след. слайде структура ПО отображена в виде последовательности слоев, где выделена отдельно **наиболее общая часть системного программного обеспечения – операционную система.**

Слои программного обеспечения



ОС как виртуальная машина. Архитектура компьютеров неудобна для использования как прикладными программистами, так и конечными пользователями. Например, работа с диском предполагает знание внутреннего устройства его электронного компонента – контроллера для ввода команд вращения диска, поиска и форматирования дорожек, чтения и записи секторов и т. д. Ясно, что прикладной программист не в состоянии учитывать все особенности работы оборудования, а должен иметь простую абстракцию, представляя информационное пространство диска как набор файлов. Файл можно открывать для чтения или записи, использовать для получения или сброса информации, а потом закрывать. Аналогичным образом скрываются от программиста все подробности работы таймера, управления памятью и т. д. Более того, на современных ЭВМ можно создать иллюзию неограниченного размера оперативной памяти и числа процессоров.

Кроме того, конечным пользователям должен быть предоставлен в распоряжение простой и удобный способ взаимодействия с компьютером через посредство терминала, что можно обеспечить только средствами ПО. **Всем этим занимается *операционная система*. Т.О., ОС представляется пользователю *виртуальной машиной*, с которой проще иметь дело, чем непосредственно с оборудованием компьютера.**

ОС как менеджер ресурсов. ОС предназначена для управления всеми частями компьютера, в т.ч. памятью, внешними устройствами, распределением времени ЦП между выполняемыми программами (т.е. *ресурсами).* Следовательно, ОС как *менеджер ресурсов* осуществляет упорядоченное и контролируемое распределение процессоров, памяти и других ресурсов между различными программами.

ОС как защитник пользователей и программ. Если ЭВМ допускает совместную работу нескольких пользователей, то возникает проблема организации их безопасной деятельности. Необходимо обеспечить сохранность информации на диске, чтобы никто не мог удалить или повредить чужие файлы. Нельзя разрешить программам одних пользователей произвольно вмешиваться в работу программ других пользователей. Нужно пресекать попытки несанкционированного использования вычислительной системы. Все это осуществляет ОС как организатор безопасной работы пользователей и их программ.

Эволюция операционных систем

Оборудование и ПО ЭВМ эволюционировали параллельно, оказывая влияние друг на друга. Появление новых технических возможностей приводило к созданию более совершенных программ, а свежие идеи в программной области стимулировали поиски новых технических решений.

Первый период (1945–1955 гг.). Все задачи организации вычислительного процесса решались вручную программистом с пульта управления. Программа загружалась в память машины с перфокарт или с помощью панели переключателей, и выполнялись строго последовательно.

ЭВМ выполняла одновременно только одну операцию (ввод-вывод или собственно вычисления). Отладка программ велась с пульта управления с помощью изучения состояния памяти и регистров машины. В целом первый период характеризуется высокой стоимостью ЭВМ, их малым количеством и низкой эффективностью использования.

Второй период (1955 г.–начало 60-х). Появились полупроводниковые элементы. Размеры компьютеров уменьшились. Снизилась стоимость их эксплуатации и обслуживания. Наблюдается бурное развитие алгоритмических языков (FORTRAN, LISP, COBOL, ALGOL-60, PL-1 и т.д.). **Появляются первые компиляторы, редакторы связей, библиотеки математических и служебных подпрограмм.** Упрощается процесс программирования. В этот период происходит разделение персонала на программистов и операторов, специалистов по эксплуатации ЭВМ.

Изменяется процесс выполнения расчетов: пользователь приносит программу с входными данными в виде колоды перфокарт ***(задания)*** и указывает необходимые ресурсы. Оператор загружает задание в память машины и запускает его на исполнение. Полученные выходные данные печатаются на принтере и пользователь получает их через некоторое время.

Смена запрошенных ресурсов вызывает приостановку выполнения программ, в результате процессор простаивает. Для повышения эффективности использования компьютера задания с похожими ресурсами начинают собирать вместе, создавая ***пакет заданий***. Появляются ***пакетные мониторы***, которые автоматизируют запуск одной программы из пакета за другой.

Третий период (начало 1960-х – 1980 г.). Произошел переход от отдельных полупроводниковых элементов типа транзисторов к интегральным микросхемам. Техника становится более надежной и дешевой. Растет сложность и количество задач, решаемых ЭВМ. Повышается производительность процессоров.

Повышению эффективности использования ЭВМ мешает низкая скорость работы механических устройств ввода-вывода. Вместо непосредственного чтения пакета заданий с перфокарт в память начинают использовать его предварительную запись на МЛ, а затем и на диск. Когда в процессе выполнения задания требуется ввод данных, они читаются с диска. Точно так же выходная информация сначала записывается на ленту или диск, а печатается только после завершения задания. Такой прием получает название **spooling** (Simultaneous Peripheral Operation On Line) или ***подкачки-откачки данных*****.** Введение техники подкачки-откачки в *пакетные системы* позволило совместить реальные операции ввода-вывода одного задания с выполнением другого задания, но потребовало обеспечения параллельной работы внешних устройств как друг с другом, так и с ЦП на основе использования аппарата ***прерываний***.

Появилось ***мультипрограммирование***: пока одна программа выполняет операцию ввода-вывода, процессор не простаивает, а выполняет другую программу. Когда операция ввода-вывода заканчивается, процессор возвращается к выполнению первой программы. Мультипрограммирование требует наличия в памяти нескольких программ одновременно. При этом каждая программа загружается в свой участок оперативной памяти - раздел, и не должна влиять на выполнение другой программы.

Внешние *прерывания* оповещают ОС о том, что произошло асинхронное событие, например завершилась операция ввода-вывода. **Внутренние *прерывания*** возникают, когда выполнение программы привело к ситуации, требующей вмешательства ОС, например деление на ноль.

МЛ были устройствами последовательного доступа. Появление МД, для которого не важен порядок чтения информации, привело к дальнейшему развитию ЭВМ. При обработке пакета заданий на МЛ очередность запуска заданий определялась порядком их ввода. При обработке пакета заданий на МД появилась возможность выбора очередного выполняемого задания. ***Пакетные системы*** **начинают заниматься планированием заданий: в зависимости от наличия запрошенных ресурсов, срочности вычислений и т.д. для обработки выбирается то или иное задание**.

Но пользователь не мог непосредственно взаимодействовать с заданием. Отладка программ по-прежнему занимала много времени и требовала изучения распечаток содержимого памяти и регистров или использования отладочной печати.

Появление терминалов на основе телетайпов, а позднее – на основе дисплеев изменили ситуацию. Расширением систем мультипрограммирования стали ***системы разделения времени*.** В них процессор переключается между задачами не только на время операций ввода-вывода, но и по прошествии определенного времени. Эти переключения происходят так часто, что пользователи могут взаимодействовать со своими программами во время их выполнения, то есть интерактивно. В результате появляется возможность одновременной работы нескольких пользователей на одной компьютерной системе.

Чтобы уменьшить ограничения на количество работающих пользователей, была внедрена идея неполного нахождения исполняемой программы в оперативной памяти: основная часть программы находится на диске, а фрагмент, который необходимо в данный момент выполнять, должен быть загружен в оперативную память, а ненужный – выгружен обратно на диск. Это реализуется с помощью механизма ***виртуальной памяти***, объем которой складывается из объема реального ОЗУ и части объема ВЗУ. Основным достоинством такого механизма является создание иллюзии неограниченной оперативной памяти ЭВМ.

Каждая программа получает свою область подобного воображаемого ОЗУ, на которой полностью помещаются как сама программа, так и все данные, которыми она оперирует. Кроме того, каждая программа получает свою элементарную порцию времени, по истечении которой по сигналу от электронных часов производится прерывание данной программы и передача управления следующей по порядку программе.

В *системах разделения времени* пользователь получил возможность эффективно производить отладку программы в интерактивном режиме и записывать информацию на диск, не используя перфокарты, а непосредственно с клавиатуры.

Этот же период времени характеризуется использованием ЭВМ для целей управления станками, процессами и т.п. в реальном времени, что обусловило создание специальных ОС ***реального времени***, ориентированных на решение задач такого рода.

Четвертый период (с 1980 г. по настоящее время). В эти годы произошло резкое возрастание степени интеграции и снижение стоимости микросхем. Компьютер по цене и простоте эксплуатации стал доступен отдельному человеку, а не отделу предприятия или университета. Наступила эра ПЭВМ.

Компьютеры стали использоваться не только специалистами, что потребовало разработки «дружественного» ПО.

В середине 80-х стали развиваться сети компьютеров, в том числе персональных, работающих под управлением ***сетевых*** или ***распределенных*** ОС.

В *сетевых* ОС пользователи могут получить доступ к ресурсам другого сетевого компьютера, если они знают об их наличии и умеют это делать. Каждая машина в сети работает под управлением своей локальной ОС, отличающейся от ОС автономного компьютера наличием дополнительных средств программной поддержки для сетевых интерфейсных устройств и доступа к удаленным ресурсам.

***Распределенная* система** внешне выглядит как обычная автономная система. Пользователь не знает, где его файлы хранятся – на локальной или удаленной машине – и где его программы выполняются. Он может вообще не знать, подключен ли его компьютер к сети.

Можно выделить шесть основных функций, которые выполняли ОС в процессе эволюции:

- планирование заданий и использования процессора;
- обеспечение программ средствами коммуникации и синхронизации;
- управление памятью;
- управление файловой системой;
- управление вводом-выводом;
- обеспечение безопасности.

Каждая из приведенных функций обычно реализована в виде подсистемы, являющейся структурным компонентом ОС.

Классификация операционных систем

ОС можно разделить на три класса: системы *пакетной обработки*, системы *разделения времени* и системы *реального времени*. При этом сетевые и распределенные ОС не являются новым классом ОС, а отражают определенные свойства систем указанных классов.

Системы пакетной обработки предназначаются для решения задач в основном вычислительного характера, не требующих быстрого получения результатов. Главной целью и критерием эффективности систем пакетной обработки является максимальная пропускная способность ЭВМ, то есть решение максимального числа задач в единицу времени.

Для достижения этой цели используются следующая схема функционирования: в начале формируется пакет заданий, каждое задание содержит требование к системным ресурсам; из этого пакета заданий формируется мультипрограммная смесь, то есть множество одновременно выполняемых задач. Для одновременного выполнения выбираются задачи, предъявляющие отличающиеся требования к ресурсам, так, чтобы обеспечивалась сбалансированная загрузка всех устройств.

Т.о., выбор нового задания из пакета заданий зависит от внутренней ситуации, складывающейся в системе, то есть выбирается «выгодное» задание. Следовательно, в таких ОС невозможно гарантировать выполнение того или иного задания в течение определенного периода времени. В них переключение процессора с выполнения одной задачи на выполнение другой происходит только в случае, если активная задача сама отказывается от процессора, например, из-за необходимости выполнить операцию ввода-вывода. Поэтому одна задача может надолго занять процессор, что делает невозможным выполнение интерактивных задач. Т.о., взаимодействие пользователя с ЭВМ сводится к тому, что он приносит задание, отдает его диспетчеру-оператору, а в конце дня после выполнения всего пакета заданий получает результат.

Системы разделения времени призваны устранить изоляцию пользователя-программиста от процесса выполнения его задач. Каждому пользователю предоставляется терминал, с которого он может вести диалог со своей программой. Так как в этих системах каждой задаче выделяется только квант процессорного времени и ни одна задача не занимает процессор надолго, то время ответа оказывается приемлемым.

Если квант выбран небольшим, то у всех пользователей, одновременно работающих на одной и той же машине, складывается впечатление, что каждый из них единолично использует машину. Ясно, что системы разделения времени обладают меньшей пропускной способностью, чем системы пакетной обработки, так как на выполнение принимается каждая запущенная пользователем задача, а не та, которая «выгодна» системе, и, кроме того, имеются накладные расходы вычислительной мощности на более частое переключение процессора с задачи на задачу. **Критерием эффективности систем разделения времени является не максимальная пропускная способность, а удобство и эффективность работы пользователя.**

Системы реального времени применяются для управления различными техническими объектами, такими, как станок, спутник, научная экспериментальная установка или технологическими процессами, такими, как гальваническая линия, доменный процесс и т.п.

Во всех этих случаях существует предельно допустимое время, в течение которого должна быть выполнена та или иная программа, управляющая объектом, в противном случае может произойти авария.

Т.о., критерием эффективности для систем реального времени является их способность выдерживать заранее заданные интервалы времени между запуском программы и получением результата (управляющего воздействия). Это время называется *временем реакции* системы, а соответствующее свойство системы - *реактивностью*.

Некоторые ОС могут совмещать в себе свойства систем разных типов, например, часть задач может выполняться в режиме пакетной обработки, а часть - в режиме реального времени или в режиме разделения времени. В таких случаях режим пакетной обработки часто называют фоновым режимом.

Основные понятия ОС

Системные вызовы. В любой ОС поддерживается механизм, который позволяет пользовательским программам обращаться к услугам ядра ОС. В ОС советской ЭВМ БЭСМ-6 - экстракоды, в ОС IBM - системные макрокоманды, в ОС Unix – *системные вызовы*.

Системные вызовы (system calls) – это интерфейс между ОС и пользовательской программой. Они создают, удаляют и используют различные объекты, главные из которых – процессы и файлы. Пользовательская программа запрашивает сервис у ОС, осуществляя *системный вызов*. В результате - *прерывание процессора (программное)*, после чего управление передается обработчику данного *вызова*, входящему в ядро ОС.

Прерывание (аппаратное) (hardware interrupt) – событие, генерируемое внешним (по отношению к процессору) устройством. Посредством *прерываний* аппаратура либо информирует ЦП о том, что произошло какое-либо событие, требующее немедленной реакции (например, пользователь нажал клавишу), либо сообщает о завершении асинхронной операции ввода-вывода (например, закончено чтение данных с диска). Важный тип *прерываний – прерывания таймера*, генерируемые периодически через фиксированный промежуток времени и используемые ОС при планировании процессов. Каждый тип аппаратных *прерываний* имеет номер, однозначно определяющий источник *прерывания*.

Исключительная ситуация (exception) – событие, возникающее в результате попытки выполнения программой команды, которая по каким-то причинам не может быть выполнена: попытки доступа к ресурсу при отсутствии достаточных привилегий или обращения к отсутствующей странице памяти, выполнение операции деления на нуль и т.п. *Исключительные ситуации*, как и *системные вызовы*, являются синхронными событиями, возникающими в контексте выполняемой программы.

Файлы. Под файлом в общем случае понимают именованную часть пространства на внешнем носителе информации, предназначенную для хранения данных. Главная задача файловой системы (file system) – скрыть особенности ввода-вывода и дать программисту простую модель файлов, независимых от устройств. Для чтения, создания, удаления, записи, открытия и закрытия файлов имеется категория *системных вызовов*.

Процессы. Понятие *процесса* тесно связано с понятием *программа.* По ходу выполнения программы компьютер обрабатывает различные команды и преобразует значения переменных. Для выполнения программы ОС должна выделить определенное количество оперативной памяти, закрепить за ней устройства ввода-вывода или файлы. Для описания выполняющейся программы и используется термин *процесс*.

Процесс можно рассматривать как абстракцию, характеризующую программу во время выполнения. Понятие *процесса* характеризует некоторую совокупность набора исполняющихся команд, ассоциированных с ним ресурсов (память, используемые файлы и устройства ввода-вывода и т. д.) и текущего момента его выполнения (значения регистров, программного счетчика, состояние стека и значения переменных), находящуюся под управлением ОС. Не существует взаимно-однозначного соответствия между процессами и программами, обрабатываемыми ЭВМ. В некоторых ОС для работы определенных программ может организовываться более одного процесса или один и тот же процесс может исполнять последовательно несколько различных программ.

Основные подсистемы операционных систем

Управление процессами

Управление памятью

Управление вводом-выводом

Управление файлами

Управление безопасностью

Управление процессами

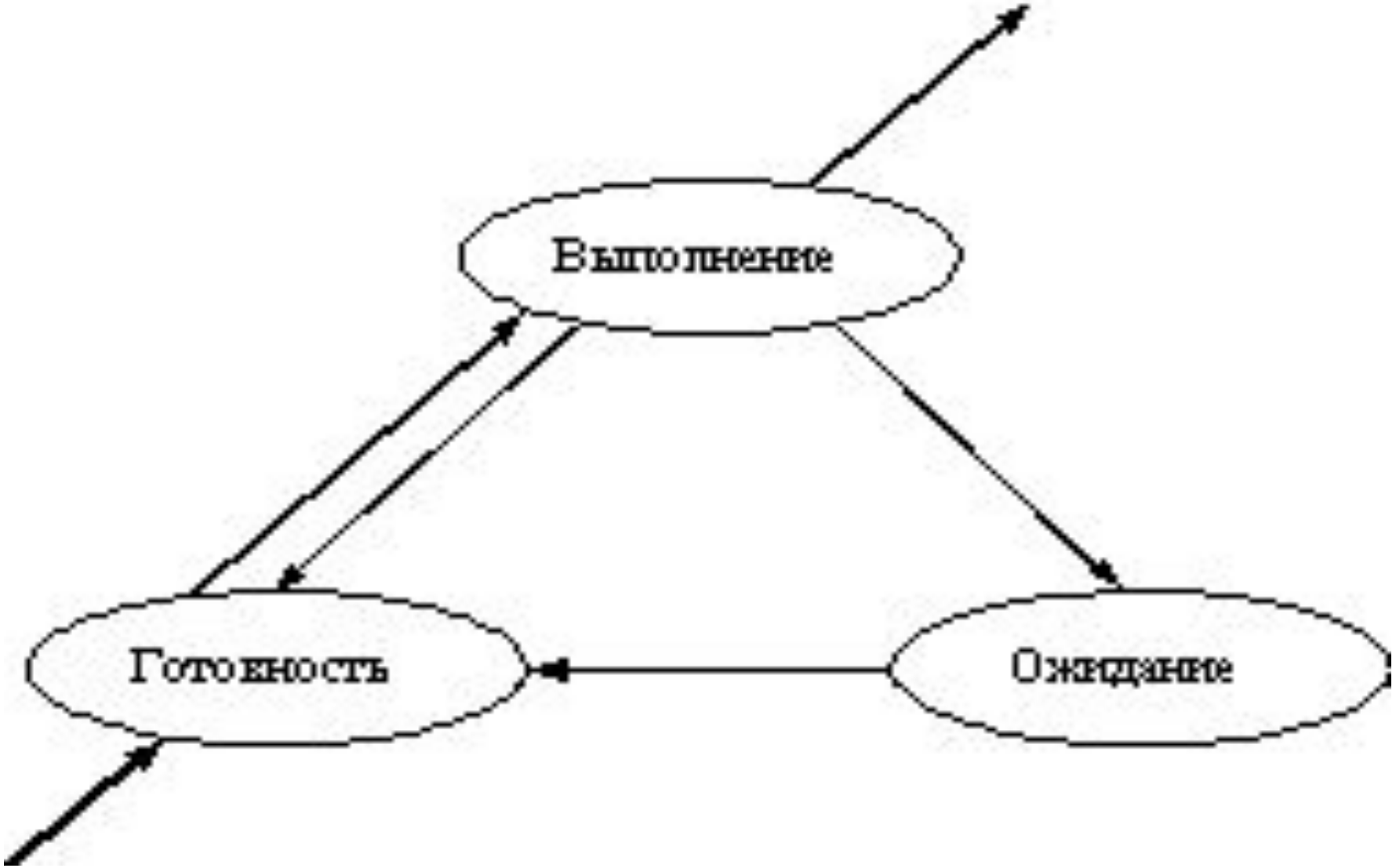
Все, что выполняется в ЭВМ (не только программы пользователей, но и определенные части ОС), организовано как набор процессов. На однопроцессорной компьютерной системе в каждый момент времени может исполняться только один процесс. Для мультипрограммных ЭВМ псевдопараллельная обработка процессов достигается с помощью переключения процессора с одного процесса на другой. Пока один процесс выполняется, остальные ждут своей очереди.

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

- **выполнение** - активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;
- **ожидание** - пассивное состояние процесса, процесс заблокирован, он не может выполняться по своим внутренним причинам, он ждет осуществления некоторого события, например, завершения операции ввода-вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса;
- **готовность** - также пассивное состояние процесса, но в этом случае он заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

В ходе жизненного цикла каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования процессов, реализуемым в данной ОС. Типичный граф состояний процесса показан на след. слайде.

Граф состояний процесса в многозадачной среде



В состоянии **Выполнение** в однопроцессорной системе может находиться только один процесс, а в каждом из состояний **Ожидание** и **Готовность** - несколько, образующих очереди ожидающих и готовых процессов. Жизненный цикл процесса начинается с состояния **Готовность**, когда процесс готов к выполнению и ждет своей очереди. При активизации процесс переходит в состояние **Выполнение** и находится в нем до тех пор, пока либо он сам освободит процессор, перейдя в состояние **Ожидание** какого-нибудь события, либо будет насильно «вытеснен» из процессора, например, вследствие исчерпания отведенного ему кванта процессорного времени. В последнем случае процесс возвращается в состояние **Готовность**. В это же состояние процесс переходит из состояния **Ожидание**, после того, как ожидаемое событие произойдет.

На протяжении существования процесса его выполнение может быть многократно прервано и продолжено. Для того, чтобы возобновить выполнение процесса, необходимо восстановить состояние его операционной среды. Состояние операционной среды отображается состоянием регистров и программного счетчика, режимом работы процессора, указателями на открытые файлы, информацией о незавершенных операциях ввода-вывода, кодами ошибок выполняемых данным процессом системных вызовов и т.д. Эта информация - **контекст процесса**. Кроме этого, ОС для планирования процессов требуются: идентификатор и состояние процесса, данные о степени его привилегированности, др. информация. Информацию такого рода называют **дескриптором процесса**. **Дескриптор процесса по сравнению с контекстом содержит более оперативную информацию, которая должна быть легко доступна подсистеме планирования процессов. Контекст процесса используется ОС только после того, как принято решение о возобновлении прерванного процесса.**

Очереди процессов - дескрипторы отдельных процессов, объединенные в списки. Т.о., каждый дескриптор содержит по крайней мере один указатель на другой дескриптор, соседствующий с ним в очереди. Такая организация очередей позволяет легко их переупорядочивать, включать и исключать процессы, переводить процессы из одного состояния в другое. Программный код только тогда начнет выполняться, когда для него ОС будет создан процесс. Создать процесс - это значит: создать информационные структуры, описывающие данный процесс, то есть его дескриптор и контекст; включить дескриптор нового процесса в очередь готовых процессов; загрузить кодовый сегмент процесса в ОП.

Управление памятью

Программы вместе с данными в процессе выполнения должны находиться в оперативной памяти. ОС приходится решать задачу распределения памяти между пользовательскими процессами и компонентами ОС. Эта деятельность – *управление памятью*.

Разновидности памяти могут быть объединены в иерархию по убыванию времени доступа, возрастанию цены и увеличению емкости.

Многоуровневую схему используют следующим образом. Информация, которая находится в памяти верхнего уровня, обычно хранится также на уровнях с большими номерами. Если процессор не обнаруживает нужную информацию на i-м уровне, он начинает искать ее на следующих уровнях. Когда нужная информация найдена, она переносится в память более высокого уровня.

Оказывается, при таком способе организации по мере снижения скорости доступа к уровню памяти снижается также и частота обращений к нему.

Память является важнейшим ресурсом, требующим тщательного управления со стороны мультипрограммной ОС. Распределению подлежит вся оперативная память, не занятая ОС.

Функциями ОС по управлению памятью являются:

отслеживание свободной и занятой памяти,

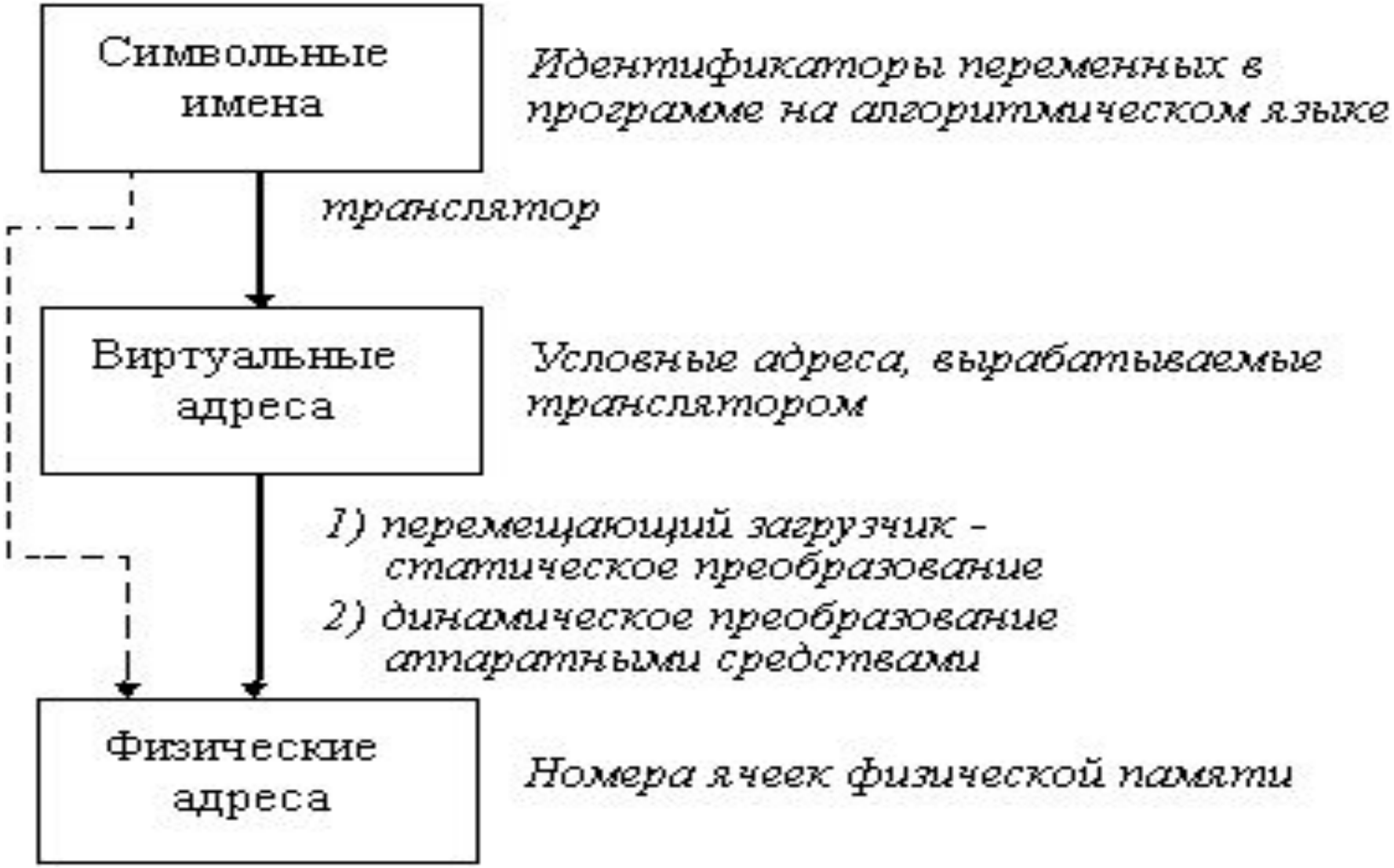
выделение памяти процессам и освобождение памяти при их завершении,

вытеснение процессов из ОП на диск, когда размеры ОП не достаточны для размещения в ней всех процессов, и возвращение их в ОП, когда в ней освобождается место,

настройка адресов программы на конкретную область физической памяти.

Типы адресов

Для идентификации переменных и команд используются *символьные* имена (метки), *виртуальные* и *физические* адреса



Символьные имена присваивает пользователь при написании программы на языке программирования.

Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык. Так как во время трансляции в общем случае не известно, в какое место оперативной памяти будет загружена программа, то транслятор присваивает переменным и командам *виртуальные* (условные) адреса, считая по умолчанию, что программа будет размещена, начиная с нулевого адреса. *Совокупность виртуальных адресов* процесса называется *виртуальным адресным пространством*. Максимальный размер виртуального адресного пространства ограничивается разрядностью адреса, присущей данной архитектуре компьютера, и, как правило, не совпадает с объемом физической памяти, имеющимся в компьютере.

Физические адреса соответствуют номерам ячеек оперативной памяти, где в действительности расположены или будут расположены переменные и команды. Переход от *виртуальных* адресов к *физическим* может осуществляться двумя способами. В первом случае замену виртуальных адресов на физические делает специальная системная программа - перемещающий загрузчик. Перемещающий загрузчик на основании имеющихся у него исходных данных о начальном адресе физической памяти, в которую предстоит загружать программу, и информации, предоставленной транслятором об адресно-зависимых константах программы, выполняет загрузку программы, совмещая ее с заменой виртуальных адресов физическими. Второй способ заключается в том, что программа загружается в память в неизмененном виде в виртуальных адресах, при этом ОС фиксирует смещение действительного расположения программного кода относительно виртуального адресного пространства. Во время выполнения программы при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический.

В некоторых случаях, когда заранее точно известно, в какой области оперативной памяти будет выполняться программа, транслятор формирует исполняемый код сразу в физических адресах.

Адреса в **основной памяти**, характеризующие реальное расположение данных в ней - *физические* адреса. Набор физических адресов, с которым работает программа, *физическое* **адресное пространство**.

Методы распределения памяти.

Все методы управления памятью могут быть разделены на два класса: методы, не использующие перемещение процессов между оперативной памятью и диском, и методы, которые делают это.

Методы распределения памяти без использования дискового пространства.

Самым простым способом управления оперативной памятью является ее предварительное (обычно на этапе генерации или в момент загрузки системы) разбиение на несколько разделов фиксированной величины (***схема с фиксированными разделами***).

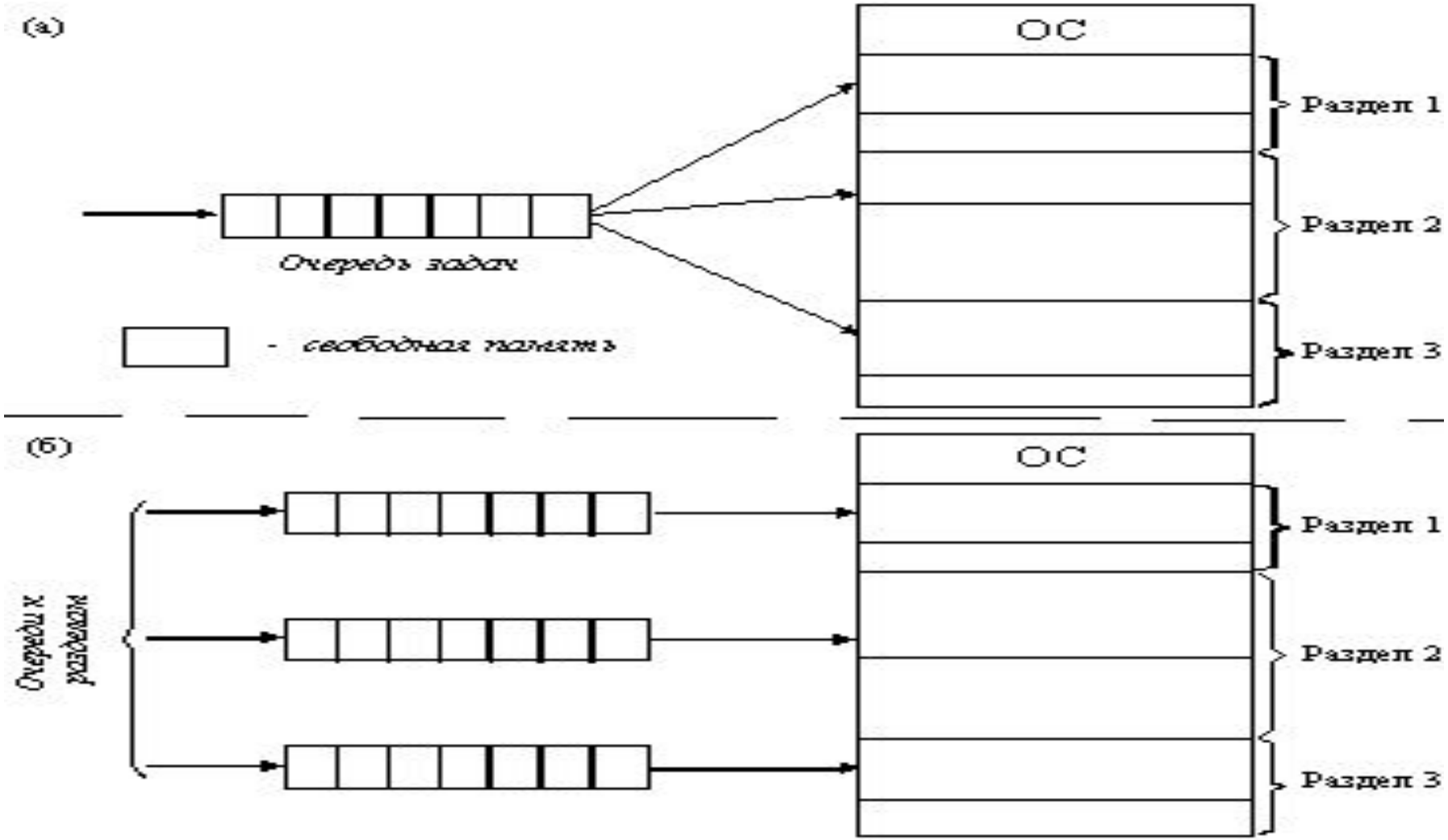
Поступающие процессы помещаются в тот или иной раздел. При этом происходит условное разбиение физического адресного пространства. Связывание логических и физических адресов процесса происходит на этапе его загрузки в конкретный раздел, иногда – на этапе компиляции.

Каждый раздел может иметь свою очередь процессов, а может существовать и глобальная очередь для всех разделов (след. слайд).

Подсистема управления памятью оценивает размер поступившего процесса, выбирает подходящий для него раздел, осуществляет загрузку процесса в этот раздел и настройку адресов.

Очевидный недостаток этой схемы – число одновременно выполняемых процессов ограничено числом разделов.

Распределение памяти фиксированными разделами
а) – с общей очередью, б) – с отдельными очередями



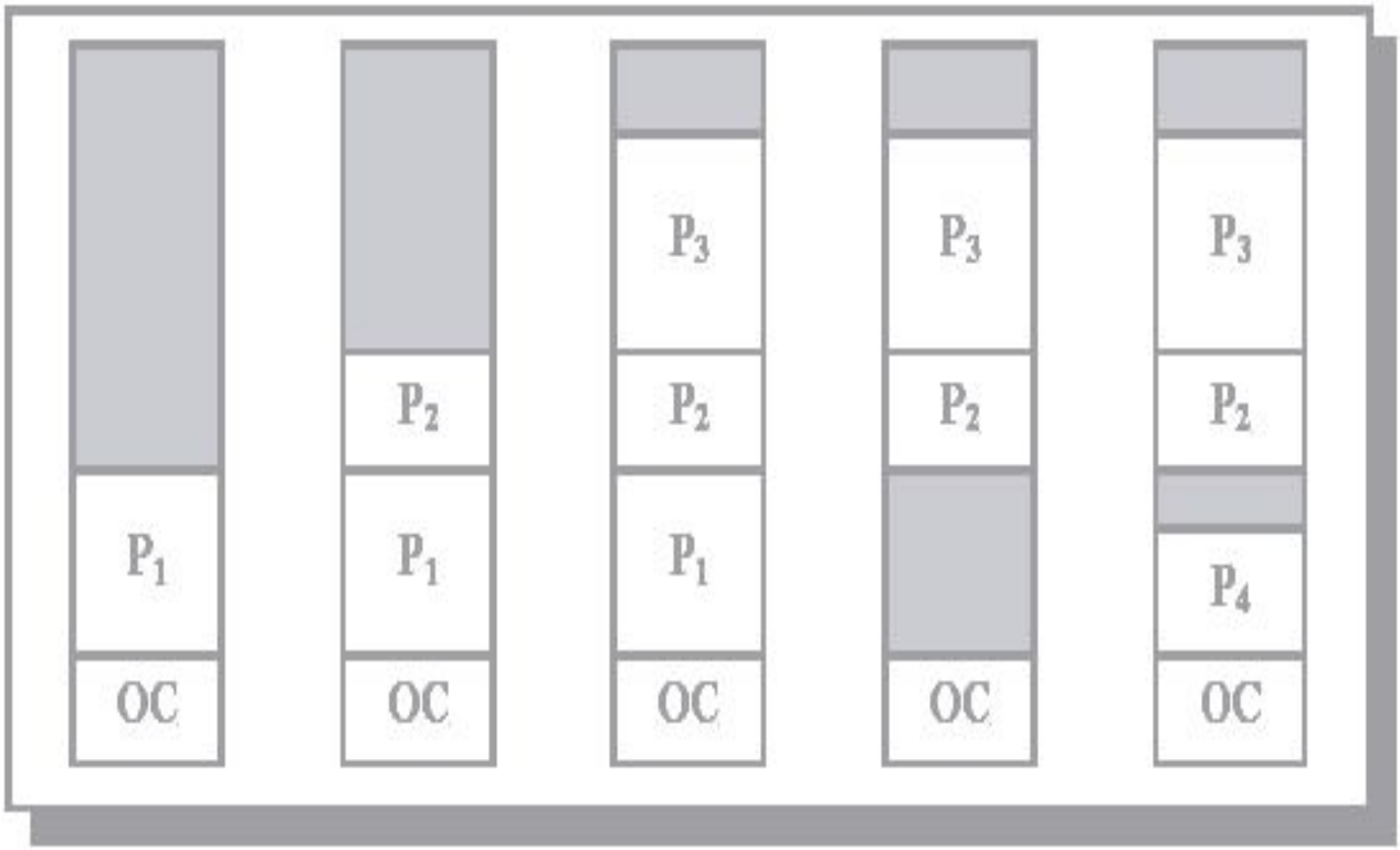
Распределение памяти разделами переменной величины. Память не делится заранее на разделы, каждой вновь поступающей программе выделяется необходимая ей память. Если достаточный объем памяти отсутствует, то программа не принимается на выполнение и стоит в очереди. После завершения программы память освобождается, и на это место может быть загружена другая программа. Т. о., в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера **(след. слайд).**

Задачами ОС является:

- ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти;
- при поступлении новой программы - анализ запроса, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения поступившей программы;
- загрузка программы в выделенный ей раздел и корректировка таблиц свободных и занятых областей;
- после завершения программы корректировка таблиц свободных и занятых областей.

Программный код не перемещается во время выполнения, то есть может быть проведена единовременная настройка адресов посредством использования перемещающего загрузчика. По сравнению с методом распределения памяти фиксированными разделами данный метод обладает гораздо большей гибкостью, но ему присущ очень серьезный недостаток - ***фрагментация памяти***. ***Фрагментация*** - это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

Динамика распределения памяти между процессами



Перемещаемые разделы. Один из методов борьбы с фрагментацией - перемещение всех занятых участков в сторону старших либо в сторону младших адресов, так, чтобы вся свободная память образовывала единую свободную область. В дополнение к функциям, которые выполняет ОС при распределении памяти переменными разделами, в данном случае она должна еще время от времени копировать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется ***сжатием***. Сжатие может выполняться либо при каждом завершении программы, либо только тогда, когда для вновь поступившей программы нет свободного раздела достаточного размера. Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то преобразование адресов из виртуальной формы в физическую должно выполняться динамическим способом.

Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может потребовать значительного времени, что часто перевешивает преимущества данного метода.

Методы распределения памяти с использованием дискового пространства..

Описанные схемы недостаточно эффективно используют память, поэтому в современных системах не принято размещать процесс в оперативной памяти одним непрерывным блоком. Вообще необходимо отметить, что проблема размещения в памяти программ, размер которых превышает имеющуюся в наличии свободную память, существовала всегда. С появлением внешней памяти на начальных этапах решением было разбиение программы на части, называемые *оверлеями*. 0-ой *оверлей* начинал выполняться первым. Когда он заканчивал свое выполнение, он вызывал другой *оверлей*. Все оверлеи хранились на диске и перемещались между памятью и диском средствами ОС. Разбиение программы на части и планирование их загрузки в оперативную память должен был осуществлять программист.

Развитие работ в этом направлении привело к появлению понятия ***виртуальной памяти***, позволяющей пользователям писать программы, размер которых превосходит имеющуюся оперативную память. Осуществляется это, как и в случае с *оверлеями*, за счет использования дискового пространства, но уже в автоматическом режиме, без участия программиста. Можно сказать, что ***виртуальная память*** – это пространство, образованное множеством *виртуальных адресов* программы, размеры которого часто превосходят размеры реальной физической оперативной памяти.

При работе с *виртуальной памятью* ОС решает следующие задачи:

- размещает данные в запоминающих устройствах разного типа, например, часть программы в оперативной памяти, а часть на диске;
- перемещает по мере необходимости данные между запоминающими устройствами разного типа, например, подгружает нужную часть программы с диска в оперативную память;
- преобразует виртуальные адреса в физические.

Наиболее распространенными реализациями *виртуальной памяти* является ***страничное***, ***сегментное*** и ***странично-сегментное*** распределение памяти, а также ***свопинг.***

Страничное распределение. На след. слайде показана схема страничного распределения памяти. Виртуальное адресное пространство каждого процесса делится на части одинакового размера, называемые ***виртуальными страницами***. Вся оперативная память машины также делится на части такого же размера, называемые ***физическими страницами*** (или *блоками*).

Размер страницы выбирается равным степени двойки: 512, 1024 и т.д., что позволяет упростить механизм преобразования виртуальных адресов в физические. При загрузке процесса часть его виртуальных страниц помещается в ОП, а остальные - на диск. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. При этом ОС создает для каждого процесса *таблицу страниц*, в которой устанавливается соответствие между номерами виртуальных и физических страниц для страниц, загруженных в оперативную память, или делается отметка о том, что виртуальная страница выгружена на диск.

При активизации очередного процесса в специальный регистр процессора загружается адрес таблицы страниц данного процесса.

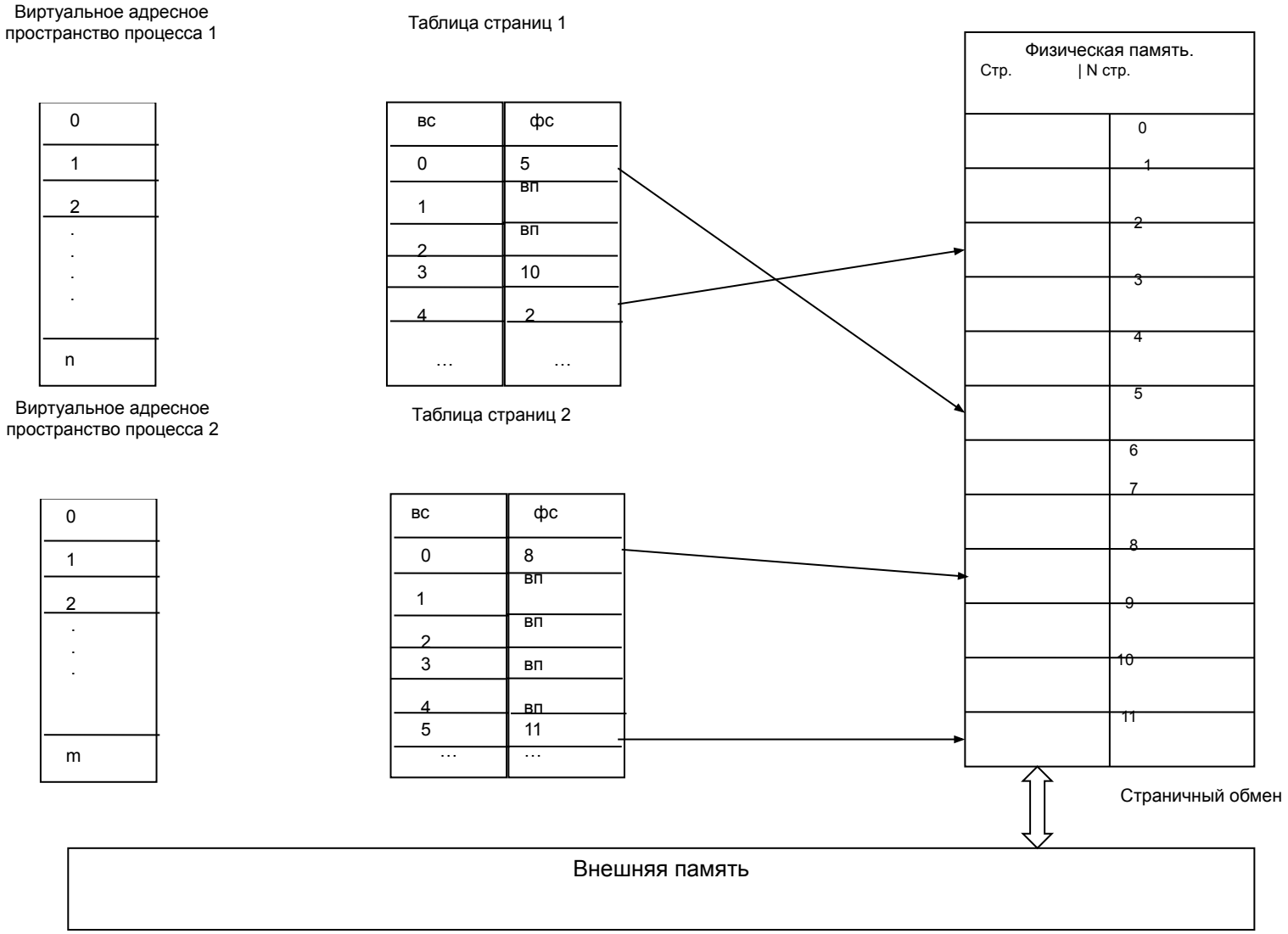
При каждом обращении к памяти происходит чтение из таблицы страниц информации о виртуальной странице, к которой произошло обращение. Если данная виртуальная страница находится в ОП, то выполняется преобразование виртуального адреса в физический.

Если нужная виртуальная страница в данный момент выгружена на диск, то происходит **страничное прерывание**. Выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди готовых. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается загрузить ее в оперативную память.

Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то решается вопрос, какую страницу следует выгрузить из оперативной памяти.

После того, как выбрана страница, анализируется ее признак модификации (из таблицы страниц). Если выталкиваемая страница с момента загрузки была модифицирована, то она должна быть переписана на диск. Если нет, то она может быть просто уничтожена.

Страничное распределение памяти ВС – виртуальная страница, ФС – физическая страница, ВП – внешняя память



Сегментное распределение. Виртуальное адресное пространство процесса делится на сегменты, размер которых определяется программистом с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т.п. Иногда сегментация программы выполняется по умолчанию компилятором.

При загрузке процесса часть сегментов помещается в ОП, а часть размещается в дисковой памяти. Сегменты одной программы могут занимать в оперативной памяти несмежные участки. Во время загрузки система создает таблицу сегментов процесса (аналогичную таблице страниц), в которой для каждого сегмента указывается начальный физический адрес сегмента в ОП, размер сегмента, правила доступа, признак модификации, признак обращения к данному сегменту за последний интервал времени и некоторая др. информация.

Если виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок ОП, в который данный сегмент загружается в единственном экземпляре.

Система с сегментной организацией функционирует аналогично системе со страничной организацией: время от времени происходят прерывания, связанные с отсутствием нужных сегментов в памяти, при необходимости освобождения памяти некоторые сегменты выгружаются, при каждом обращении к ОП выполняется преобразование виртуального адреса в физический

Недостатком данного метода распределения памяти является фрагментация на уровне сегментов и более медленное по сравнению со страничной организацией преобразование адреса.

Странично-сегментное распределение. Хранить в памяти сегменты большого размера целиком так же неудобно, как и хранить процесс непрерывным блоком. Напрашивается идея разбиения сегментов на страницы. В этом случае виртуальное пространство процесса делится на сегменты, а каждый сегмент в свою очередь делится на виртуальные страницы, которые нумеруются в пределах сегмента. Оперативная память делится на физические страницы.

Загрузка процесса выполняется ОС постранично, при этом часть страниц размещается в оперативной памяти, а часть - на диске.

Для каждого сегмента создается своя таблица страниц, структура которой полностью совпадает со структурой таблицы страниц, используемой при страничном распределении.

Для каждого процесса создается таблица сегментов, в которой указываются адреса таблиц страниц для всех сегментов данного процесса. Адрес таблицы сегментов загружается в специальный регистр процессора, когда активизируется соответствующий процесс.

***Свопинг* - разновидность виртуальной памяти.** В соответствии с этим методом некоторые процессы (обычно находящиеся в состоянии ожидания) временно выгружаются на диск. ОС не исключает их из своего рассмотрения и при наступлении условий активизации некоторого процесса, находящегося в области свопинга на диске, этот процесс перемещается в оперативную память. Если свободного места в оперативной памяти не хватает, то выгружается другой процесс.

При свопинге, в отличие от рассмотренных ранее методов реализации виртуальной памяти, процесс перемещается между памятью и диском целиком, то есть в течение некоторого времени процесс может полностью отсутствовать в оперативной памяти.

Существуют различные алгоритмы выбора процессов на загрузку и выгрузку, а также различные способы выделения оперативной и дисковой памяти загружаемому процессу.

УПРАВЛЕНИЕ ВВОДОМ ВЫВОДОМ

Функционирование любой вычислительной системы сводится к выполнению двух видов работ:

обработке информации и операций по осуществлению ее ввода-вывода.

Одной из главных функций ОС при этом является управление устройствами ввода-вывода компьютера (УВВ).

ОС должна передавать устройствам команды, перехватывать прерывания и обрабатывать ошибки; обеспечивать интерфейс между устройствами и остальной частью системы.

Физические принципы организации ввода-вывода. Устройства ввода-вывода делятся на два типа: *блок-ориентированные* и *байт-ориентированные*.

Блок-ориентированные - хранят информацию в блоках фиксированного размера, каждый из которых имеет свой собственный адрес. Пример – магнитный диск.

Байт-ориентированные - не адресуемы, не позволяют производить операцию поиска, они генерируют или потребляют последовательность байтов. Примеры - терминалы, строчные принтеры, др.

Некоторые ВУ не относятся ни к одному классу, например, часы, которые не адресуемы и не порождают потока байтов. Это устройство только выдает сигнал прерывания в некоторые моменты времени. ВУ обычно состоит из механического и электронного компонента. Электронный компонент - *контроллер* устройства или адаптер. Механический компонент - собственно устройство. Некоторые контроллеры могут управлять несколькими устройствами. **ОС имеет дело не с устройством, а с контроллером.** Контроллер, как правило, выполняет простые функции, например, преобразует поток бит в блоки, состоящие из байт, и осуществляют контроль и исправление ошибок. Каждый контроллер имеет несколько регистров, которые используются для взаимодействия с центральным процессором. В некоторых компьютерах эти регистры являются частью физического адресного пространства. В таких компьютерах нет специальных операций ввода-вывода. В других компьютерах адреса регистров ввода-вывода, называемых *портами*, образуют собственное адресное пространство за счет введения специальных операций ввода-вывода.

ОС выполняет ввод-вывод, записывая команды в регистры контроллера. При завершении команды контроллер организует прерывание для того, чтобы передать управление процессором ОС, которая должна проверить результаты операции. Процессор получает результаты и статус устройства, читая информацию из регистров контроллера.

Несмотря на все многообразие устройств, управление их работой и обмен информацией с ними строятся на небольшом наборе принципов, которые мы и рассмотрим ниже.

Общие сведения об архитектуре современного компьютера. Процессор, память и ВУ связаны большим количеством электрических соединений – линий, которые в совокупности называют **локальной магистралью компьютера**. Внутри локальной магистрали линии, предназначенные для выполнения сходных функций, группируют в ***шины***. В современных компьютерах выделяют как минимум три шины:

- **шину данных,** состоящую из линий данных и служащую для передачи информации между процессором и памятью, процессором и устройствами ввода-вывода, памятью и ВУ;
- **адресную шину,** состоящую из линий адреса и служащую для задания адреса ячейки памяти или указания устройства ввода-вывода, участвующих в обмене информацией;
- **шину управления**, состоящую из линий управления локальной магистралью и линий ее состояния, определяющих поведение локальной магистрали.

Количество линий, входящих в состав шины - ее разрядность (ширина). Ширина адресной шины определяет максимальный размер оперативной памяти, которая может быть установлена в вычислительной системе. Ширина шины данных определяет максимальный объем информации, которая за один раз может быть получена или передана по этой шине.

ВУ могут подключаться к локальной магистрали в одной или множестве точек - ***портов ввода-вывода***. Каждый порт имеет свой номер или адрес в ***адресном пространстве портов ввода-вывода.***

Когда адресное пространство оперативной памяти задействовано не полностью (остались адреса, которым не соответствуют физические ячейки памяти), часть портов может быть отображена в адресное пространство памяти (например, видеопамять дисплеев). В этом случае действия, необходимые для обмена данными с ВУ не отличаются от действий, производимых для передачи информации между оперативной памятью и процессором, и для их выполнения применяются те же самые команды.

Если порт отображен в адресное пространство портов ввода-вывода, то процесс обмена информацией инициируется командами ввода-вывода. Что должны делать устройства, приняв информацию через свой порт, определяется ***контроллерами.***

Общие черты контроллеров. Обычно каждый контроллер имеет по крайней мере четыре внутренних регистра, называемых регистрами **состояния**, **управления**, **входных данных** и **выходных данных**. Для доступа к содержимому этих регистров вычислительная система может использовать один или несколько портов. Для простоты будем считать, что каждому регистру соответствует свой порт.

Регистр состояния содержит биты, значение которых определяется состоянием устройства ввода-вывода и которые доступны только для чтения вычислительной системой. Эти биты - бит занятости, бит готовности данных, бит ошибки и т. д.

Регистр управления получает данные для инициализации устройства ввода-вывода или выполнения очередной команды, а также для изменения режима работы устройства.

Регистр выходных данных служит для помещения в него данных для чтения вычислительной системой.

Регистр входных данных предназначен для помещения в него информации, которая должна быть выведена на устройство.

Рассмотрим, как выполняется команда вывода данных с использованием контроллера:

- процессор читает информацию из регистра состояний и проверяет значение бита занятости. Если бит занятости установлен, то это означает, что устройство еще не завершило предыдущую операцию, и процессор повторяет операцию чтения. Если бит занятости сброшен, то устройство готово к выполнению новой операции, и процессор переходит на следующий шаг;
- процессор записывает код команды вывода в регистр управления;
- процессор записывает данные в регистр входных данных;
- процессор устанавливает бит готовности команды. **Далее процессор не участвует!**
- когда контроллер определяет, что бит готовности команды установлен, он устанавливает бит занятости;
- контроллер анализирует код команды в регистре управления и обнаруживает, что это команда вывода. Он берет данные из регистра входных данных и инициирует выполнение команды;
- после завершения операции контроллер обнуляет бит готовности команды;
- при успешном завершении операции контроллер обнуляет бит ошибки в регистре состояния, при неудачном завершении команды – устанавливает его;
- контроллер сбрасывает бит занятости.

Как видим, процессор ожидает освобождения устройства, непрерывно опрашивая значение бита занятости. Такой способ взаимодействия процессора и контроллера получил название ***способа опроса устройств.***

Для того, чтобы процессор не дожидался состояния готовности устройства ввода-вывода, непрерывно опрашивая его, а мог выполнять в это время другую работу, необходимо, чтобы устройство само умело сигнализировать процессору о своей готовности. Механизм, который позволяет ВУ оповещать процессор о завершении команды вывода или команды ввода - ***механизм внешних прерываний*.**

В простейшем случае для реализации механизма прерываний необходимо к имеющимся шинам локальной магистрали добавить линию, соединяющую процессор и УВВ – линию прерываний. По завершении выполнения операции ВУ выставляет на эту линию специальный сигнал, по которому процессор после выполнения очередной команды сохраняет содержимое своих регистров и переходит на выполнение *программы обработки* прерывания, расположенной по определенному адресу. При наличии одной линии прерываний процессор при этом должен опросить состояние всех УВВ, чтобы определить, от какого из них пришло прерывание и выполнить необходимые действия.

В современных ЭВМ устройства сообщают о своей готовности процессору через *контроллер прерываний*, при этом используется не одна линия, а целая шина прерываний. Каждому устройству присваивается свой номер, который при возникновении прерывания контроллер передает процессору. Этот номер служит индексом в *таблице* прерываний, содержащей адреса программ обработки прерываний – *векторы* прерываний.

Не все ВУ являются одинаково важными с точки зрения ЭВМ: некоторые прерывания являются более существенными, чем другие. Контроллер прерываний позволяет устанавливать *приоритеты* для прерываний от ВУ. При одновременном возникновении прерываний от нескольких устройств процессору сообщается номер наиболее приоритетного из них для его обслуживания в первую очередь. Менее приоритетное прерывание при этом не пропадает. При обработке прерывания процессор может получить сообщение о возникновении прерывания с более высоким приоритетом и переключиться на его обработку. Похожим образом процессор обрабатывает исключительные ситуации и программные прерывания (*внутренние прерывания*), возникающие во время выполнения процессором команды.

Использование механизма прерываний позволяет загружать процессор в то время, когда УВВ занимается своей работой. Однако запись или чтение большого количества информации приводят к большому количеству операций ввода-вывода, которые должен выполнять процессор. Для освобождения процессора был предложен механизм прямого доступа внешних устройств к памяти – ПДП или Direct Memory Access – DMA. Выполняется такой доступ через специальный *контроллер прямого доступа к памяти*, который и управляет такого рода обменом без участия центрального процессора (исключая этапы инициализации и окончания обмена).

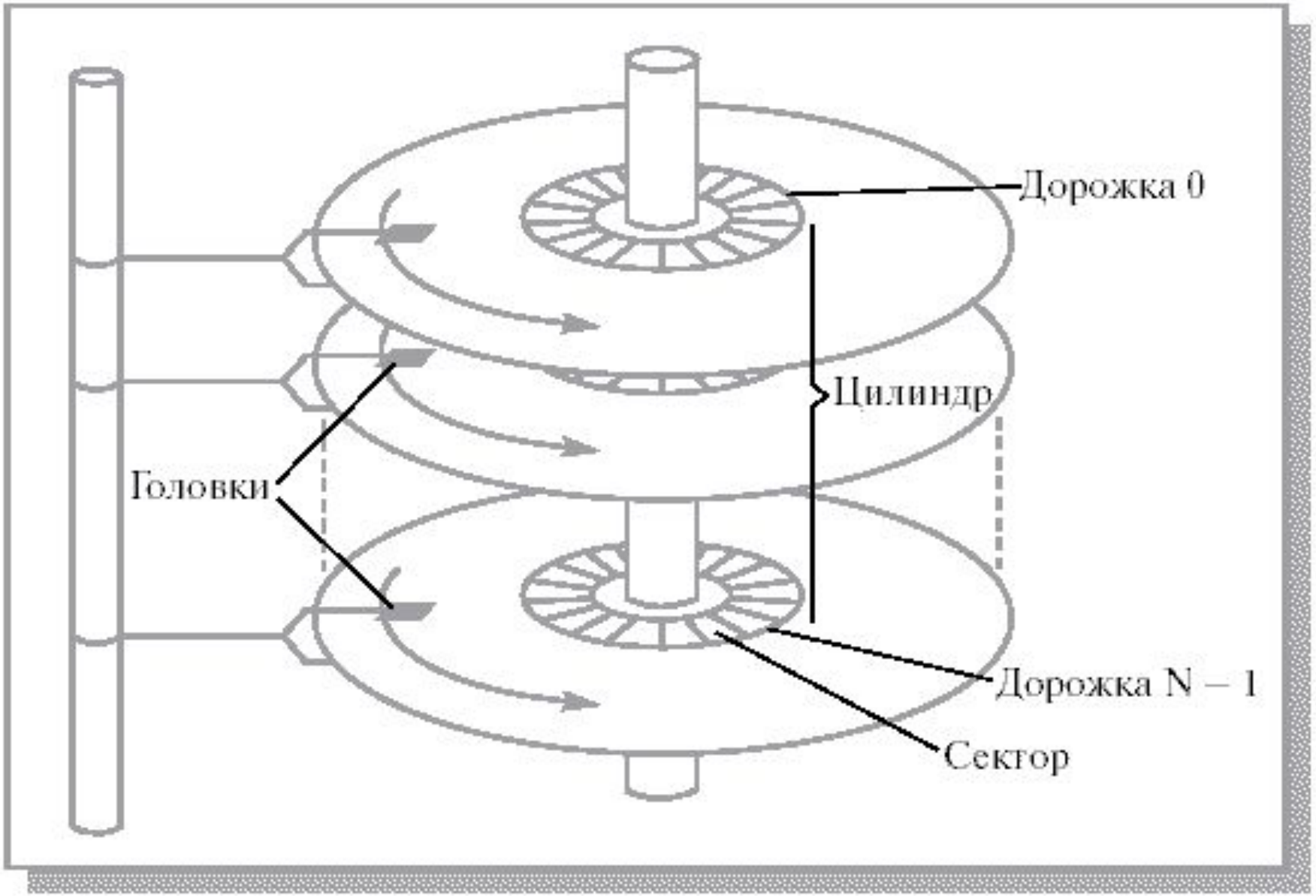
Строение жесткого диска. Современный жесткий МД представляет собой набор круглых пластин, находящихся на одной оси и покрытых с одной или двух сторон специальным магнитным слоем (след. слайд). Около каждой рабочей поверхности каждой пластины расположены магнитные головки для чтения и записи информации.

Эти головки присоединены к специальному рычагу, который может перемещать весь блок головок над поверхностями пластин как единое целое. Поверхности пластин разделены на концентрические кольца, внутри которых и может храниться информация. Набор концентрических колец на всех пластинах для одного положения головок (т. е. все кольца, равноудаленные от оси) образует ***цилиндр***. Каждое кольцо внутри цилиндра получило название ***дорожки*** (по одной или две дорожки на каждую пластину). Все дорожки делятся на равное число ***секторов***.

Количество дорожек, цилиндров и секторов может варьироваться от одного жесткого диска к другому в широких пределах. Как правило, сектор является минимальным объемом информации, которая может быть прочитана с диска за один раз.

При работе диска набор пластин вращается вокруг своей оси с высокой скоростью, подставляя по очереди под головки соответствующих дорожек все их сектора. Номер сектора, номер дорожки и номер цилиндра однозначно определяют положение данных на жестком диске и, наряду с типом совершаемой операции – чтение или запись, полностью характеризуют часть запроса, связанную с устройством, при обмене информацией в объеме одного сектора.

Схема жесткого диска



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

Ключевым принципом является независимость от устройств. Вид программы не должен зависеть от того, читает она данные с гибкого или с жесткого диска. В частности, для именования устройств должны быть приняты единые правила.

Другим важным вопросом является обработка ошибок. Вообще говоря, ошибки следует обрабатывать как можно ближе к аппаратуре. Если контроллер обнаруживает ошибку чтения, то он должен попытаться ее скорректировать. Если же это не удается, то исправлением ошибок должен заняться *драйвер* устройства. И только если нижний уровень не может справиться с ошибкой, он сообщает об ошибке верхнему уровню.

Еще один ключевой вопрос - использование блокирующих (**синхронных**) и неблокирующих (**асинхронных**) передач. Большинство операций физического ввода-вывода выполняется асинхронно - процессор начинает передачу и переходит на другую работу, пока не наступает прерывание. Пользовательские программы легче писать, если операции ввода-вывода блокирующие - после команды READ программа приостанавливается до тех пор, пока данные не попадут в ее буфер. ОС выполняет операции ввода-вывода асинхронно, но представляет их для пользовательских программ в синхронной форме.

Последняя проблема - одни устройства являются **разделяемыми**, а другие - **выделенными**. Диски - разделяемые устройства, т.к. к нему возможен одновременный доступ нескольких пользователей. Принтеры - выделенные устройства, потому что нельзя смешивать строчки, печатаемые различными пользователями.

Для решения данных проблем делят ПО ввода-вывода на четыре слоя (след. слайд):

обработка прерываний; драйверы устройств; независимый от устройств слой ОС; пользовательский слой ПО.

Многоуровневая организация подсистемы ввода-вывода

Обработка прерываний. Прерывания должны быть скрыты как можно глубже в недрах ОС, чтобы как можно меньшая часть ОС имела с ними дело. Наилучший способ - разрешить процессу, инициировавшему операцию ввода-вывода, блокировать себя до завершения операции и наступления прерывания. При наступлении прерывания процедура обработки прерывания выполняет разблокирование процесса, инициировавшего операцию ввода-вывода.

Драйверы устройств. Весь зависимый от устройства код помещается в драйвер устройства, являющийся частью ОС. Каждый драйвер управляет устройствами одного типа или одного класса.

В ОС только драйвер устройства знает о его конкретных особенностях. Например, только драйвер диска имеет дело с дорожками, секторами, цилиндрами и др. факторами, обеспечивающими правильную работу диска. Драйвер устройства принимает запрос от программного слоя и решает, как его выполнить. Типичным запросом является чтение *n* блоков данных. Если драйвер был свободен во время поступления запроса, то он начинает выполнять запрос немедленно. Если же он был занят обслуживанием другого запроса, то вновь поступивший запрос присоединяется к очереди уже имеющихся запросов, и он будет выполнен, когда наступит его очередь.

Первый шаг в реализации запроса ввода-вывода, например, для диска, состоит в преобразовании его из абстрактной формы в конкретную. Для дискового драйвера это означает преобразование номеров блоков в номера цилиндров, головок, секторов, проверку, работает ли мотор, находится ли головка над нужным цилиндром.

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

Независимый от устройств слой операционной системы. Большая часть ПО ввода-вывода является независимой от устройств. Типичными функциями для независимого от устройств слоя являются:

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

Остановимся на некоторых функциях данного перечня.

Обеспечение независимого размера блока. Верхним слоям ПО неудобно работать с блоками разной величины, поэтому данный слой обеспечивает единый размер блока, например, за счет объединения нескольких различных блоков в единый логический блок. В связи с этим верхние уровни имеют дело с абстрактными устройствами, которые используют единый размер логического блока независимо от размера физического сектора.

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

Буферизация. Буфер - некоторая область памяти для запоминания информации при обмене данными между двумя устройствами, двумя процессами или процессом и устройством. Нас будет интересовать использование буферов в том случае, когда одним из участников обмена является внешнее устройство. Есть четыре причины использования буферов в подсистеме ввода-вывода.

Разные скорости приема и передачи информации, которыми обладают участники обмена. Рассмотрим случай передачи потока данных от клавиатуры к модему. Скорость, с которой поставляет информацию клавиатура, определяется скоростью набора текста человеком и существенно меньше скорости передачи данных модемом. Для того чтобы не занимать модем на все время набора текста, делая его недоступным для других процессов и устройств, целесообразно накапливать введенную информацию в буфере и отсылать ее через модем после его заполнения.

Разные объемы данных, которые могут быть приняты или получены участниками обмена единовременно. Возьмем другой пример. Пусть информация поставляется модемом и записывается на диск. Помимо обладания разными скоростями совершения операций, модем и диск представляют собой устройства разного типа. Модем является символьным устройством и выдает данные байт за байтом, в то время как диск является блочным устройством и для проведения операции записи для него требуется накопить необходимый блок данных в буфере.

Необходимость копирования данных из приложений, осуществляющих ввод-вывод, в специальную область оперативной памяти, откуда они и будут выведены на внешнее устройство по мере освобождения последнего (в противном случае при использовании асинхронных обменов в пользовательской программе данные, подлежащие выводу на внешнее устройство, могут подвергнуться изменению еще до выполнения операции вывода) .

Необходимость сокращения времени обмена между оперативной памятью и внешним устройством за счет использования механизма кэширования (с понятием кэш-памяти мы сталкивались при рассмотрении иерархии памяти).

Пользовательский слой программного обеспечения. Хотя большая часть программного обеспечения ввода-вывода находится внутри ОС, некоторая его часть содержится в библиотеках, связываемых с пользовательскими программами.

Системные вызовы, включающие вызовы ввода-вывода, обычно делаются библиотечными процедурами. Набор подобных процедур является частью системы ввода-вывода. В частности, форматирование ввода или вывода выполняется библиотечными процедурами. Стандартная библиотека ввода-вывода содержит большое число процедур, которые выполняют ввод-вывод и работают как часть пользовательской программы.

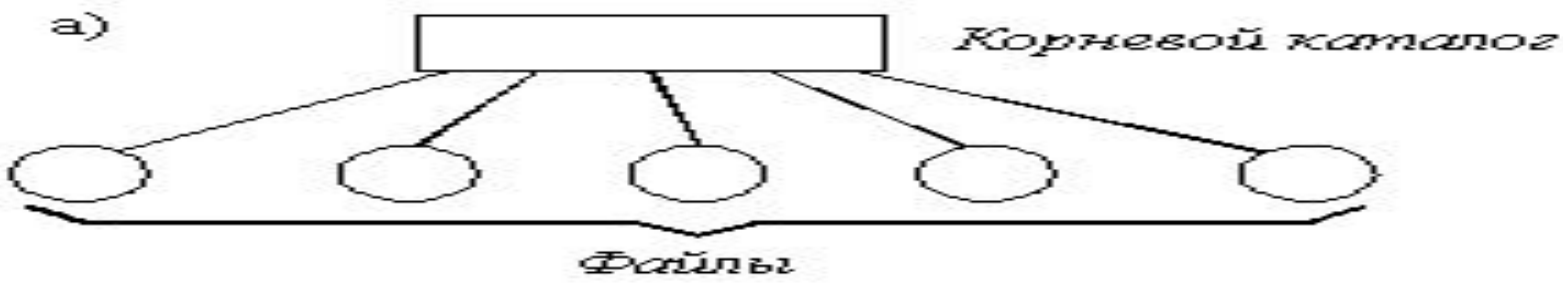
Другой категорией ПО ввода-вывода является подсистема **спулинга (spooling**). **Спулинг** - это способ работы с выделенными устройствами в мультипрограммной системе.

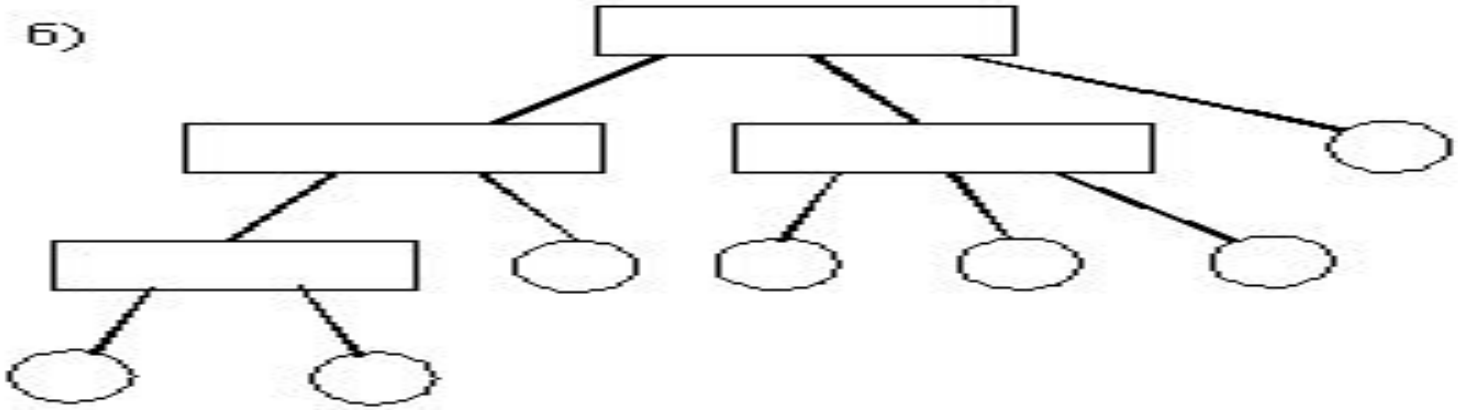
Типичное устройство, требующее спулинга - строчный принтер. Создается специальный процесс - ***монитор***, который получает исключительные права на использование этого устройства. Также создается специальный каталог, называемый ***каталогом спулинга***. Для того, чтобы напечатать файл, пользовательский процесс помещает выводимую информацию в этот файл и помещает его в каталог спулинга. Процесс-монитор по очереди распечатывает все файлы, содержащиеся в каталоге спулинга.

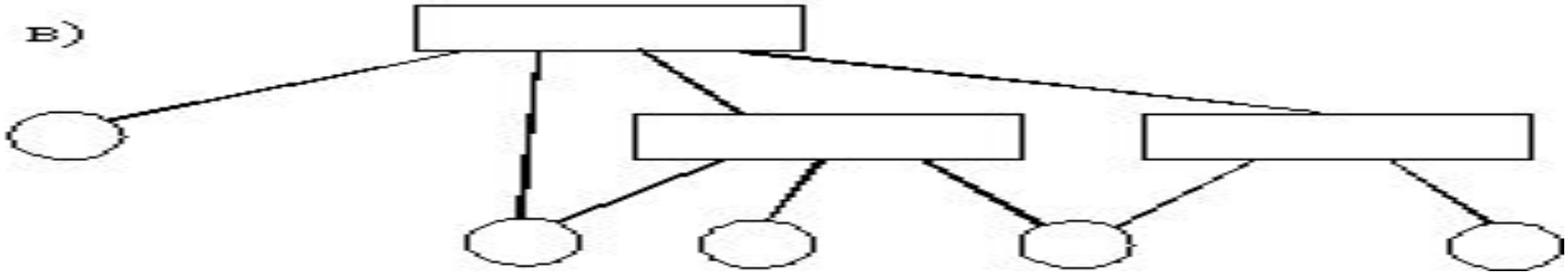
Управление файлами

Организация файловой системы

а) - одноуровневая; б) - иерархическая (дерево); в) - иерархическая (сеть)

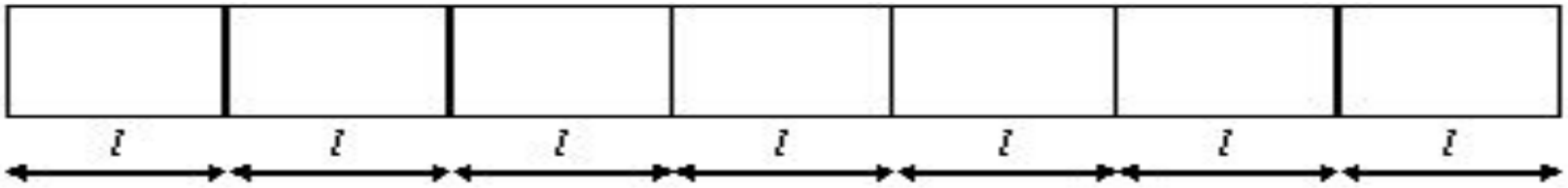






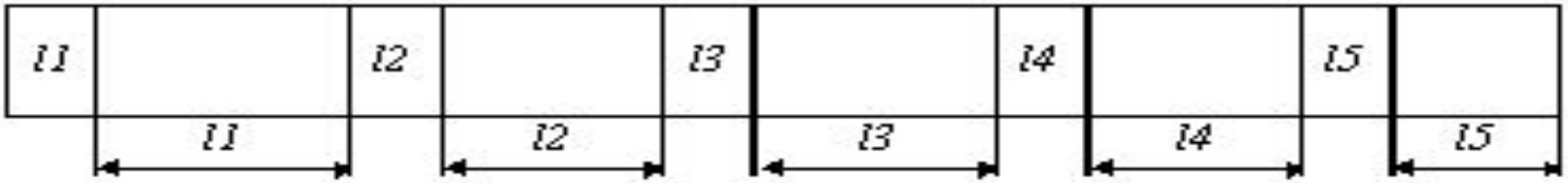
Способы логической организации файлов

I



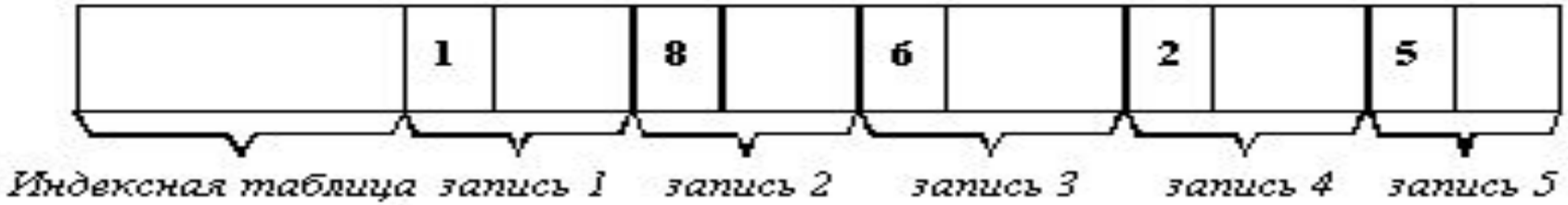
Последовательность логических записей фиксированной длины

II



Последовательность логических записей переменной длины

III



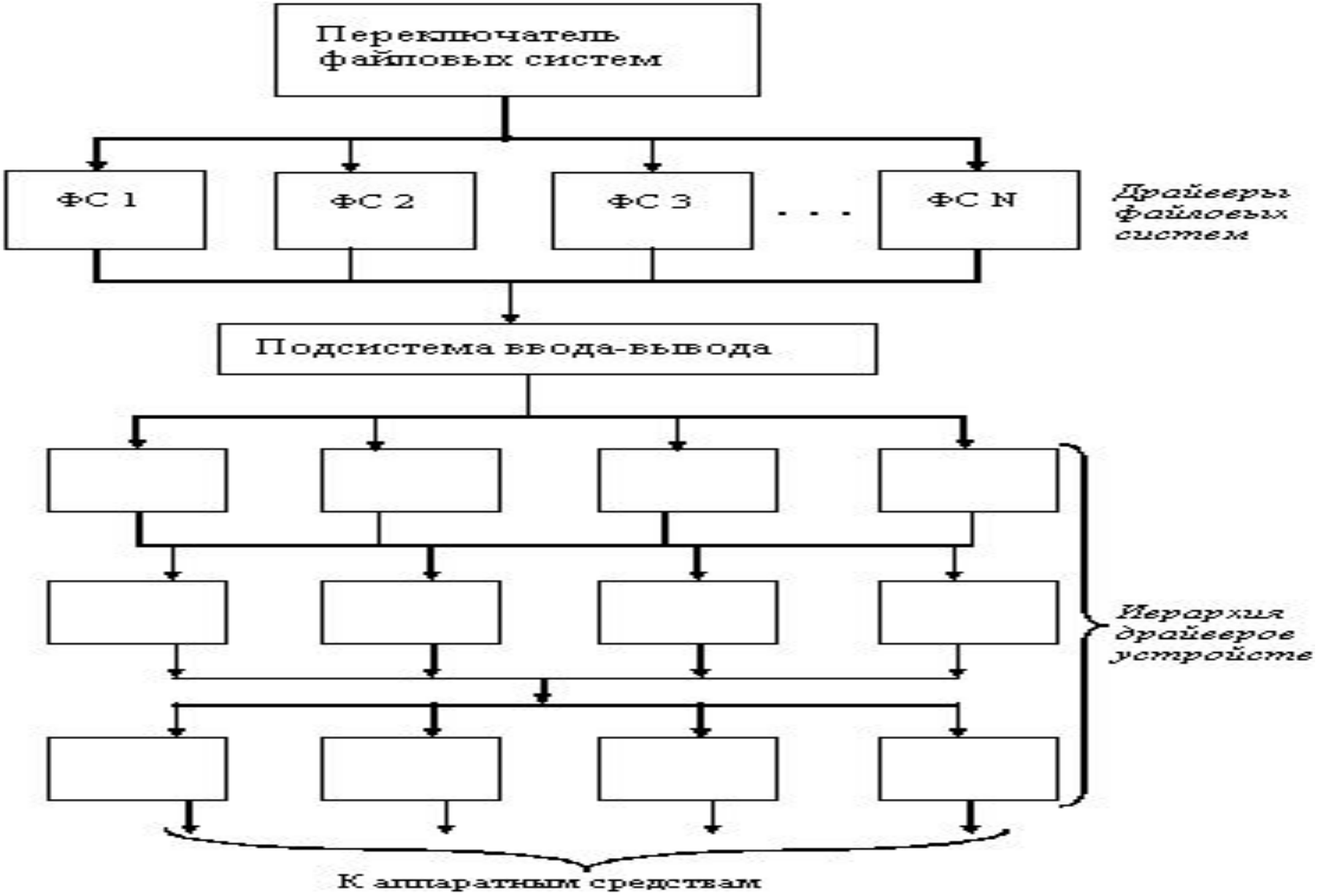
Индексная логическая организация

Индекс	1	2	3	4	5	6
Адрес	21	201	315	661	670	715

Индекс ≡ ключ

Общая модель файловой системы

Архитектура современной файловой системы



ЗАЩИТНЫЕ МЕХАНИЗМЫ ОПЕРАЦИОННЫХ СИСТЕМ

Основные задачи системы защиты ОС:

идентификация;

аутентификация;

авторизация;

протоколирование;

аудит.

Идентификация и аутентификация.

Наиболее распространенным способом контроля доступа являются процедуры **идентификации** и **последующей аутентификации** пользователя. **Идентификация** основана на использовании регистрационного имени пользователя (идентификатора), позволяющего системе определить его атрибуты, заданные при первоначальной регистрации в ОС. Для того, чтобы установить, что пользователь именно тот, за кого себя выдает, используется процедура **аутентификации,** задача которой - предотвращение доступа к системе нежелательных лиц.

Для аутентификации используются: ключ или магнитная карта; секретный пароль; отпечатки пальцев, подпись, голос, др.

Пароли. Когда пользователь идентифицирован системой, у него запрашивается пароль. Если пароль, сообщенный пользователем, совпадает с паролем, хранящимся в системе, система предполагает, что процедура аутентификации закончена.

Недостатки паролей связаны с тем, что они могут быть угаданы, случайно показаны или нелегально переданы одним пользователем другому.

Есть несколько основных способов выяснить чужой пароль. Один связан со сбором информации о пользователе (в качестве паролей часто используют имена, даты рождения, номерные знаки автомобилей, пр.), другой основан на переборе всех наиболее вероятных комбинаций букв, чисел и знаков пунктуации (атака по словарю). Кроме того, может быть просто похищена база данных о пользователях, в которой хранятся зашифрованные пароли, а затем применен специальный инструмент для их расшифровки. В некоторых случаях можно взломать пароль, получив информацию из пакетов запрос/ответ протокола аутентификации, перехваченных в сети.

Чтобы заставить пользователя выбрать наиболее сложный пароль, во многих системах используется процедура, которая при помощи собственной программы-взломщика может оценить качество пароля, введенного пользователем.

Авторизация - определение полномочий и уровня доступа пользователя к ресурсам.

После успешно выполненных процедур идентификации и аутентификации система должна предоставить пользователю права на доступ к ее ресурсам, определенные ранее администратором ОС. При этом система контроля доступа базируется на общей модели, называемой **матрицей доступа**. Рассмотрим ее более подробно.

Компьютерная система может быть представлена как набор субъектов (процессы, пользователи) и объектов. Под объектами мы понимаем как ресурсы оборудования (процессор, сегменты памяти, принтер, диски и ленты), так и программные ресурсы (файлы, программы), то есть все то, доступ к чему контролируется. Каждый объект имеет уникальное имя, отличающее его от других объектов в системе, и каждый из них может быть доступен через определенные операции.

Операции зависят от объектов. Например, процессор может только выполнять команды, сегменты памяти могут быть записаны и прочитаны, считыватель магнитных карт может только читать, а файлы данных могут быть записаны, прочитаны, переименованы и т. д.

Различают **дискреционный (*избирательный*)** способ управления доступом и **полномочный (*мандатный*).**

При дискреционном доступе определенные операции над конкретным ресурсом запрещаются или разрешаются субъектам. **Текущее состояние прав доступа при дискреционном управлении описывается матрицей, в строках которой перечислены субъекты, в столбцах - объекты, а в ячейках - операции, которые субъект может выполнить над объектом.**

Полномочный подход заключается в том, что все объекты могут иметь уровни секретности, а все субъекты делятся на группы, образующие иерархию в соответствии с уровнем допуска к информации. **Иногда это называют моделью многоуровневой безопасности.**

Домены безопасности.

Введем понятие ***домена безопасности***. Каждый домен определяет набор объектов и типов операций, которые могут производиться над каждым объектом. Возможность выполнять операции над объектом есть права доступа, каждое из которых есть упорядоченная пара <имя объекта, вид доступа>. Домен, таким образом, есть набор прав доступа. Например, если домен D имеет права доступа <file F, {read, write}>, это означает, что процесс, выполняемый в домене D, может читать или писать в файл F, но не может выполнять других операций над этим объектом. Пример доменов можно увидеть на след. слайде.

Связь конкретных субъектов, функционирующих в ОС, может быть организована следующим образом:

- **каждому пользователю может соответствовать домен.** В этом случае набор объектов, к которым может быть организован доступ, зависит от результатов идентификации пользователя;
- **каждому процессу может соответствовать домен.** В этом случае набор доступных объектов определяется идентификацией процесса;
- **каждой процедуре может соответствовать домен.** В этом случае набор доступных объектов соответствует локальным переменным, определенным внутри процедуры.

Матрица доступа. Список доменов безопасности может быть представлен в виде матрицы, называемой матрицей доступа. Эту матрицу можно разложить по столбцам, в результате чего получим *списки разрешенных прав доступа к каждому объекту для каждого домена*. В результате разложения матрицы по строкам получаем *списки разрешенных прав доступа ко всем объектам для каждого домена*, т.е. характеристики соответствующего *мандата*.

ДОМЕНЫ БЕЗОПАСНОСТИ

Объект / Домен	F1	F2	F3	Printer
D1	read			
D2				print
D3		read	execute	
D4	read write		read write	

Выявление вторжений. Аудит системы защиты.

Основным инструментом выявления вторжений является сохранение данных о работе ОС. Отдельные действия пользователей протоколируются, а полученный протокол используется для выявления вторжений.

Аудит, т.о., заключается в регистрации специальных данных о различных типах событий, происходящих в системе и так или иначе влияющих на состояние безопасности компьютерной системы. К числу таких событий обычно относят следующие: вход или выход из системы; выполнение операций над файлами (открытие, закрытие, переименование, удаление, пр.); обращение к удаленной системе; смена привилегий или иных атрибутов безопасности (режима доступа, уровня благонадежности пользователя и т. п.).

Если фиксировать все события в системе, объем регистрационной информации будет расти слишком быстро, а ее эффективный анализ станет невозможным. Обычно предусматривают наличие средств выборочного протоколирования как в отношении пользователей, так и в отношении событий.

Помимо протоколирования, можно периодически сканировать систему на наличие слабых мест в системе безопасности. Такого рода сканирование может проверить разнообразные аспекты системы (наличие легко раскрываемых паролей, изменения в системных программах, обнаруженные