

**Современный
Гуманитарный
Университет**

Дистанционное образование

Рабочий учебник

Фамилия, имя, отчество _____

Факультет _____

Номер контракта _____

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

ЮНИТА 1

ОПЕРАЦИОННЫЕ СИСТЕМЫ

МОСКВА 2000

Разработано В.Н. Кузубовым

Рекомендовано Министерством
общего и профессионального
образования Российской Федерации
в качестве учебного пособия для
студентов высших учебных заведений

КУРС: СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Юнита 1. Операционные системы.

Юнита 2. Системное программное обеспечение процессов разработки и сопровождение программных комплексов.

Юнита 3. Системное программное обеспечение сетей ЭВМ.

ЮНИТА 1

Излагаются вопросы системного программного обеспечения. Основой, к которой привязано изложение всей юниты является понятие (схема) иерархии сред – от уровня логических схем, до прикладных программ. Рассмотрены состав, структура и основные виды системного ПО – их назначение и место в иерархии.

Кроме того, в юните детально рассматриваются вопросы, связанные с развитием операционных систем – от пакетной обработки к сетевым ОС, обеспечивающим режим “клиент-сервер”. Подробно разбираются методы управления сетевыми ресурсами и организация режима многозадачной работы.

Уделено значительное внимание вопросам преемственности при переходе от одной версии системного ПО к другой.

Для студентов Современного Гуманитарного Университета

Юнита соответствует профессиональной образовательной программе № 1

(С) СОВРЕМЕННЫЙ ГУМАНИТАРНЫЙ УНИВЕРСИТЕТ, 2000

ОГЛАВЛЕНИЕ

ДИДАКТИЧЕСКИЙ ПЛАН	4
ЛИТЕРАТУРА	5
ТЕМАТИЧЕСКИЙ ОБЗОР	6
1. Назначение и состав системного ПО	6
1.1. Назначение системного ПО	6
1.2. Операционные системы	8
1.2.1. Введение в операционные системы	8
1.2.2. Операционные системы клиент-сервер	11
1.3. Системное ПО разработки ПС	16
1.4. Системное ПО разработки АИС	23
1.5. Интерфейсы и оболочки	26
1.6. Системное ПО телекоммуникационных сетей	32
2. Основы операционных систем	34
2.1. Развитие, назначение и структура операционных систем	34
2.1.1. Основные понятия	34
2.1.2. Операционная система OS/360	38
2.1.3. ДОС – операционная система для персональных ЭВМ	50
2.1.4. Система прерываний персональных ЭВМ	53
2.1.5. Режим многозадачности	55
2.1.6. Драйверы операционной системы	60
2.1.7. Операционная система UNIX	68
2.2. Управление системными ресурсами	72
2.3. Управление вводом-выводом	75
2.4. Управление внешней памятью и файловые системы	78
ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ	84
ГЛОССАРИЙ*	

* Глоссарий расположен в середине учебного пособия и предназначен для самостоятельного заучивания новых понятий.

ДИДАКТИЧЕСКИЙ ПЛАН

Назначение и состав системного ПО. Назначение системного ПО. Операционные системы. Введение в операционные системы. Операционные системы клиент-сервер. Системное ПО разработки программных средств. Системное ПО разработки АИС. Интерфейсы и оболочки. Системное ПО телекоммуникационных сетей. Основы операционных систем. Развитие, назначение и структура операционных систем. Основные понятия ОС. Операционная система OS/360. Операционная система ДОС. Система прерываний персональных ЭВМ. Режим многозадачности. Драйверы операционной системы. Операционная система UNIX. Управление системными ресурсами. Управление вводом-выводом. Управление внешней памятью и файловые системы.

ЛИТЕРАТУРА

Базовая

1. Фигурнов В.Э. IBM PC для пользователя. М., 1996.

Дополнительная

2. Windows NT – выбор “профи”. М., 1996.
3. Ресурсы Microsoft Windows 95. В 2-х т. М., 1996.
4. Липаев В.В. Управление разработкой программных средств: методы, стандарты, технология. М., 1993.

Примечание. Тематический обзор составлен с использованием всех источников.

Современный Гуманитарный Университет

ТЕМАТИЧЕСКИЙ ОБЗОР*

1. НАЗНАЧЕНИЕ И СОСТАВ СИСТЕМНОГО ПО

1.1. Назначение системного ПО

Системное программное обеспечение – это все программные средства, необходимые для реализации прикладной системы (автоматизированной информационной системы – АИС, или отдельных прикладных программ), удовлетворяющей требованиям конечных пользователей.

Под **системным программированием** понимается работа, выполняемая системными программистами, т.е. создание системного программного обеспечения. *Граница между системным программированием и прикладным программированием зависит от обстоятельств.* Например, с точки зрения программиста, который занимается ядром операционной системы, человек, создающий компилятор, является пользователем системы (т. е. прикладным программистом), хотя этот же человек будет, вероятно, рассматриваться как системный программист тем лицом, которое занимается написанием программ для отыскания минимума функции. Вместе с тем, человек, который будет использовать программу минимизации, *может считать всех трех вышеупомянутых программистов системными программистами.*

В данном курсе под **системным программистом** будет пониматься человек, который специализируется по системному программированию и программному обеспечению нижнего уровня, т.е. операционным системам, компиляторам, системам связи и системам управления базами данных и др.

Развитие системного программного обеспечения происходило параллельно с развитием аппаратных средств вычислительной техники и связи.

В середине 50-х гг. в основном разрабатывались прикладные программы, реализующие функции, для которых были необходимы возможности компьютеров. Производители технических средств изготавливали компьютеры, а пользователи разрабатывали программы с помощью операционных систем и ассемблеров, находящихся в самом начале своего развития. Режим выполнения программ зависел не от примитивных операционных систем (ОС), а от набора команд и архитектуры ЭВМ. В конце 50-х – начале 60-х гг. начали появляться управляющие системы, ориентированные на определенные типы ЭВМ. Эти программные средства, вначале состоявшие из программ ввода-вывода, являлись ресурсом коллективного пользования. Они управляли

* Жирным шрифтом выделены новые понятия, которые необходимо усвоить. Знание этих понятий будет проверяться при тестировании.

вводом-выводом на перфокарты, печатающие устройства, магнитные ленты, а к концу этого периода – на устройства памяти прямого доступа (магнитные барабаны и диски).

Дополнительно появились языки высокого уровня, такие, как Фортран и Кобол, которые существенно упростили разработку программ. Эффект состоял в значительном увеличении производительности прикладных программистов. Однако, по большей части, прикладные системы оставались архитектурно зависимы от аппаратных средств.

В середине – конце 60-х гг. резко возросла значимость операционных систем. Это произошло в связи с появлением новых возможностей, таких, как большая оперативная память, и увеличением сложности оборудования. В результате появился класс программных систем, включающих языки высокого уровня и ряд других средств. Этот класс систем явился большим шагом вперед, поскольку разрабатываемые на языках высокого уровня прикладные системы прямо не зависели от особенностей оборудования.

Но сложные прикладные системы по-прежнему требовали для увеличения их эффективности использования машинно-ориентированных средств и поэтому продолжали зависеть от архитектуры ЭВМ. Разработкой средств программирования, совершенствованием операционных систем, обеспечением пакетного режима и автоматического планирования заданий и другими возможностями управления разделяемыми ресурсами закончился первый большой этап развития операционных систем.

В 70-х гг. стали возможными интерактивный доступ к вычислительным ресурсам с использованием терминалов, а также средства межмашинной коммуникации. Как следствие, операционные системы были пополнены соответствующими функциями. Это, в свою очередь, позволило решить проблемы создания распределенных систем реального времени для коммерческих приложений. Ведущими технологиями этого, второго, этапа были обработка баз данных и передача информации. Эти средства совместно со средствами разработки интерактивных систем предоставили возможности управления компьютерными системами в режиме ON-LINE (неавтономный режим работы пользователя в сетях ЭВМ).

Были созданы надлежащие условия для эффективной разработки прикладных систем. Имеются в виду средства управления данными, представленными в реляционной модели, генераторы прикладных программ, средства управления дисплеями и пр. В результате прикладные программы были отделены от операционных систем и оборудования. При использовании этой новой, *ориентированной на приложения технологии основные силы тратятся на решение проблемы, а не на реализацию в среде конкретной ЭВМ* или операционной системы.

Описываемые изменения затронули большинство семейств ЭВМ. В середине 70-х гг. каждое семейство компьютеров с соответствующими

операционной и прикладными системами было ориентировано на конкретные приложения с учетом таких критериев, как производительность и среда применения (имеются в виду пакетный и интерактивный режимы или режим обработки транзакций). Каждый продукт разрабатывался так, чтобы оказаться лучшим среди других подобных продуктов. Такая стратегия оказалась успешной применительно к условиям рыночной конкуренции.

В 80-х гг. сильно возросла мощность всего ряда ЭВМ (от ПЭВМ до самых больших ЭВМ). Чтобы устоять в условиях сильной конкуренции, были добавлены новые возможности. Сегодня во всех системах мощность оборудования и программные средства достигли в своем развитии такой точки, когда прикладные системы должны обеспечивать только функциональный уровень и практически не зависят от аппаратных средств, хотя остаются частично зависимыми от архитектуры операционной системы, под управлением которой они выполняются.

В последующий период времени большое влияние на развитие системного ПО оказали средства организации сетей (Systems Network Architecture – SNA), которые позволили достаточно эффективно управлять большими вычислительными сетями. Интерактивные средства управления вычислениями, объединенные с возможностями построения вычислительных сетей, обеспечили мощные средства управления автоматизированными системами в режиме реального времени.

В целом, структуру системного ПО, сложившуюся к настоящему времени, можно изобразить схемой, представленной на рис. 1.

Эта модель программного обеспечения может рассматриваться как организационное средство производства прикладных систем. С помощью модели не только проще классифицировать программные продукты, разработанные в среде разных операционных систем, но и обеспечить базис для определения общих для всех ОС интерфейсов, которые помогут унифицировать разработку приложений.

Далее в данной главе будет дана краткая характеристика современного состояния основных компонент системного программного обеспечения.

1.2. Операционные системы

1.2.1. Введение в операционные системы

Операционные системы осуществляют управление системными ресурсами с целью эффективного, безопасного и надежного выполнения прикладных программ. В основе любой ОС лежит ядро операционной системы, управляющей процессами операционной системы и распределяющей для них системные ресурсы. Ядро ОС располагается постоянно в оперативной памяти ЭВМ.

При включении питания ЭВМ системный загрузчик (компонента

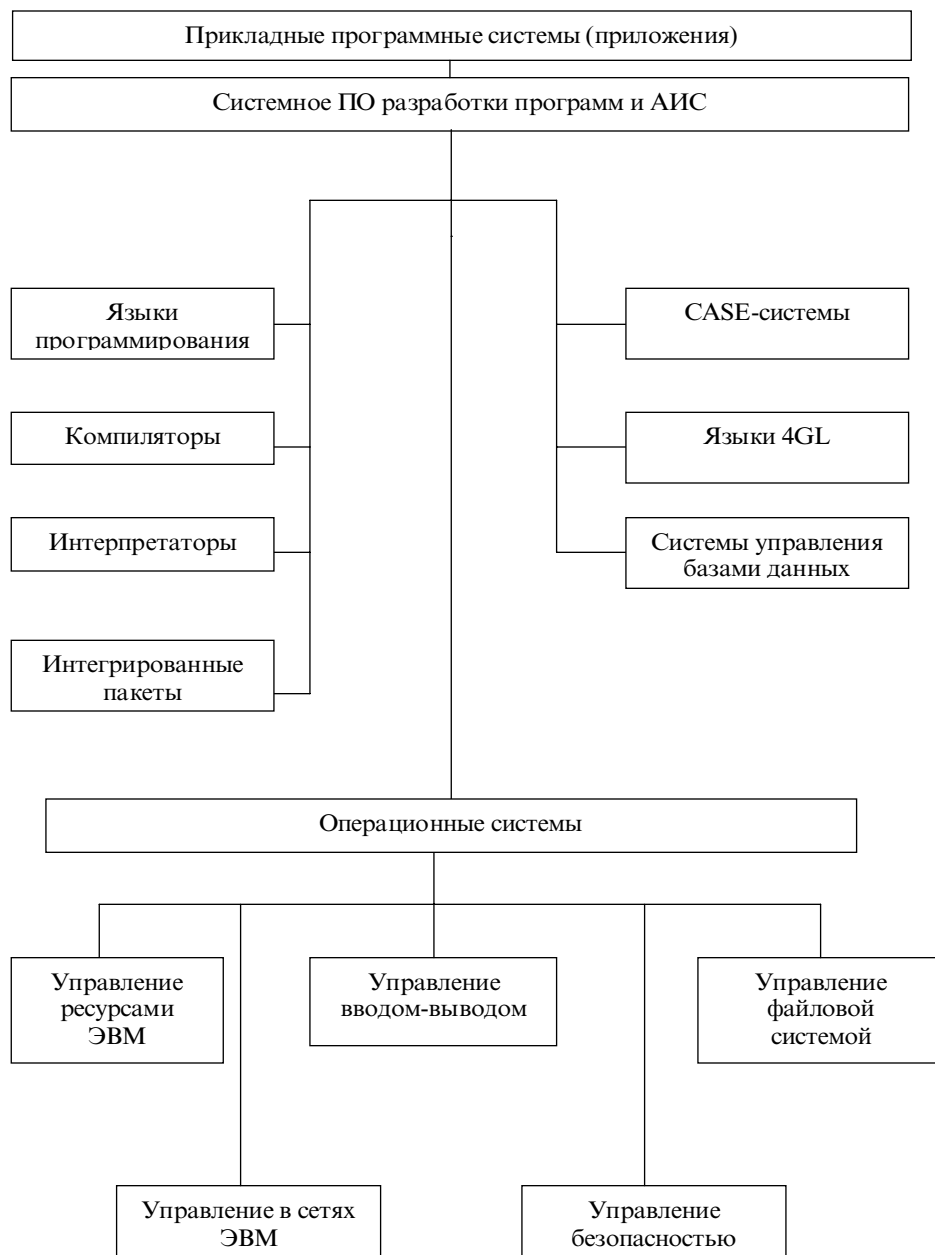


Рис. 1. Общая схема системного программного обеспечения

управляющей программы) осуществляет начальную загрузку ядра операционной системы и его инициализацию, после чего все управление системными процессами осуществляет ОС. Взаимодействие между ОС и всеми остальными программно-техническими компонентами АИС осуществляется через систему прерываний – комплекс технических и программных средств, обеспечивающих возможность прерывать выполнение программы при поступлении внешних или внутренних сигналов прерывания и продолжить выполнение прерванной программы от точки прерывания после обработки сигналов прерывания.

ОС состоит из набора системных программ, входящих в состав операционной системы и выполняющих управляющие и обслуживающие функции. Системные программы размещаются в системной библиотеке, содержащей определенные компоненты операционной системы. Каждый ресурс имеет уникальное имя, идентифицирующее его в данной операционной системе. Все события, происходящие в управляемой ОС системе, регистрируются в системном журнале (специальном наборе данных).

Известны следующие виды ОС:

- локальные операционные системы, осуществляющие управление ресурсами в пределах конкретного вычислительного комплекса, без использования каналов связи;
- сетевые операционные системы, осуществляющие управление ресурсами в локальных и глобальных сетях передачи данных.

Кроме того, могут использоваться специальные ОС, например операционные системы автоматизированного проектирования и операционные системы реального времени. Система реального времени – это система, осуществляющая информационный обмен с другими системами, периферийными устройствами или датчиками при таких временных характеристиках, которые позволяют немедленно обрабатывать всю поступающую информацию и информацию для вывода. Эти системы обычно используются в управлении техническими объектами (АСУ ТП).

Значительная доля современных автоматизированных систем имеет распределенный характер, то есть их компоненты распределены по различным вычислительным комплексам, которые связаны между собой каналами передачи данных – средствами двустороннего обмена данными, представляющими собой совокупность аппаратуры окончания канала данных и линии передачи данных. Используется также термин *передающая среда* – совокупность линии передачи данных и, возможно, коммутаторов данных, концентраторов данных, повторителей и другого оборудования, не относящегося к станции данных, организация структуры и функционирование которой обеспечивают физическую передачу данных между станциями данных. Распределенные системы базируются на сетях ЭВМ – совокупности сети передачи данных, взаимосвязанных сетью ЭВМ и необходимых для реализации этой связи

программного обеспечения и технических средств, которая предназначена для организации распределенной обработки информации.

Логическая структура и принципы работы информационной сети называются архитектурой сети, для описания которой используются следующие основные понятия:

передача данных – пересылка данных при помощи средств связи из одного места для приема их в другом;

функциональный блок сети – система или устройство, выполняющие определенную, логически связанную группу функций;

источник данных – функциональный блок, порождающий данные для передачи;

приемник данных – функциональный блок, принимающий переданные данные;

протокол – набор семантических и синтаксических правил, определяющий поведение функциональных блоков при передаче данных;

обмен данными – передача данных в соответствии с установленным протоколом;

соединение – связь, устанавливаемая между функциональными блоками для передачи информации;

ресурсы сети – программное, техническое, информационное и организационное обеспечение информационной сети, предназначенное для решения прикладных задач.

В настоящее время сетевые программные системы создаются на базе концепции открытых систем. Открытая система – это совокупность систем, взаимодействие которых подчиняется требованиям стандартов Международной организации по стандартизации. Совокупность технических и (или) программных средств, входящая в состав открытой системы, реализующая согласованные по горизонтали и вертикали функции некоторого уровня и всех расположенных ниже уровней базовой эталонной модели и выполняющая функции поставщика сервиса для компонентов сети, называется *службой взаимосвязи открытых систем*.

1.2.2. Операционные системы клиент-сервер

Разновидностью открытых систем являются системы, реализованные на базе сетей типа “intranet” – корпоративных сетей, созданных на базе технологий Интернет.

В принципе все продукты и решения, разработанные для Интернет, могут использоваться и для интрасетей. Но, тем не менее, в их реализации есть некоторые особенности, которые объясняют выделение технологии создания интрасетей в некоторое самостоятельное направление и, соответственно, появление для них некоторых специфических программных решений. Интернет представляет собой сеть из компьютеров-узлов, взаимодействие между которыми выполняется с помощью протоколов TCP/IP. На физическом уровне узлы

обычно связаны в некоторые внутренние сети, поэтому Интернет фактически является иерархической структурой: “Интернет-сети-узлы”. Но, с логической точки зрения, каждый узел является самостоятельной единицей, имеющей свой уникальный IP (Internet Protocol) адрес, что обеспечивает возможность общения любых двух узлов Интернет. В основе организации Интернет лежат две основные идеи.

Во-первых, взаимодействие узлов не должно зависеть от типов компьютеров, их архитектур, операционных систем и пр., а также физической реализации связи между ними. Для осуществления этого используются простые и надежные протоколы установления связи и передачи данных. Причем вопрос о том, насколько узел понимает смысл принимаемых или передаваемых данных, является его личной проблемой.

Во-вторых, Интернет должен надежно функционировать независимо от изменения своей топологии, в частности, от подключения или отключения его узлов. Причина обеспечения такой надежности заключается в том, что этот процесс никак не контролируется. Единственное, за чем здесь осуществляется контроль – это за обеспечением уникальности IP адресов узлов и соблюдением определенного порядка в их нумерации для решения проблем маршрутизации.

Логика развития корпоративных компьютерных сетей привела к объединению локальных сетей, реализованных, условно говоря, на уровне подразделения (здания), в глобальные. Однако создание интрасетей на уровне прямого взаимодействия компьютеров через Интернет является просто невозможным. Во-первых, это может быть просто нецелесообразно для относительно небольших сетей, в которых объединение компьютеров выполняется за счет чисто внутренних линий (например, на территории завода). Во-вторых (и это самое главное), из-за необходимости обеспечения информационной защиты при взаимодействии с внешним миром – ведь открытые протоколы Интернет делают прозрачными все включенные в него узлы. Специфика программных продуктов и технологий для интрасетей заключается в том, что число узлов в них является ограниченным и изменение топологии сети и содержимого информации носит некоторый предсказуемый характер. А в этом случае организация таких процедур, как, например, маршрутизация и поиск информации может быть оптимизирована совершенно другими методами, чем в хаотичном Интернет.

При создании информационных систем на базе интрасетей необходимо решить следующие задачи:

- полностью интегрировать внутренние, локальные сети с Интернет для расширения возможностей коммуникации предприятий с клиентами и партнерами;
- реализовать новые методы навигации, применяемые в Интернет, во всех продуктах, с тем, чтобы облегчить операции поиска и анализа информации, а также взаимодействия с партнерами;

- упростить разработку, внедрение и администрирование прикладных программ с целью оптимизации и сокращения циклов разработки;
- интегрировать новые продукты и Интернет-технологии с существующими инфраструктурами, чтобы пользователи могли на основе уже имеющихся решений постепенно развивать свои информационные системы.

В основе решения всех этих задач лежат сетевые операционные системы. Напомним, что **операционная система** – это совокупность программ, предназначенная для обеспечения определенного уровня эффективности вычислительной системы за счет автоматизированного управления ее работой и предоставляемого пользователям определенного набора услуг.

Наибольший объем современных сетевых ОС представлен классами клиентских и серверных операционных систем, что определяется быстрым развитием сетей ЭВМ.

Клиентская ОС начала 2000-х гг. – это надежная, устойчивая и мощная операционная система для персональных компьютеров, которая обладает высоким уровнем защиты данных, работает с различными файловыми системами, на различных аппаратных платформах и поддерживает многопроцессорные системы. Как правило, эта ОС локализована, т.е. имеет полностью русифицированный интерфейс – все позиции меню, система справки, системные утилиты и приложения, входящие в состав, переведены на русский язык. Локализованная версия корректно поддерживает кириллицу на системном уровне и позволяет использовать русскоязычные приложения.

Сетевые возможности клиентской ОС позволяют использовать ее в качестве клиента сетей ЭВМ, а иногда, как невыделенный сервер, в одноранговых сетях, т.е. управлять правами доступа к отдельным каталогам и файлам. Пользователь может контролировать исполнение приложений, системных служб, просматривать динамический график использования памяти и процессора.

Важное значение придается безопасности и надежности. Приложения изолируются друг от друга и от аппаратуры. Тем самым исключено влияние некорректно работающего приложения на другие и работоспособность системы в целом. Многозадачный режим дает пользователям возможность работать быстро и выполнять одновременно несколько приложений без потери производительности. Например, пользователь может работать с офисными приложениями в то время, когда в фоновом режиме идет загрузка данных из Интернет и печать документа.

Все клиентские ОС включают в свой состав средства для работы с Интернет:

- средство для навигации в Интернете;
- инструменты создания персональных Web-страниц и распространения информации внутри корпоративных сетей.

Клиентские ОС позволяют выбирать при загрузке определенную конфигурацию, с которой будет работать конкретный пользователь. Профиль, который выбирается при загрузке, включает настройки разрешения экрана, информацию о запускаемых сервисах и устройствах. Администратор рабочей станции может создавать профили пользователей, в которых содержатся настройки рабочего стола и окружения для каждого пользователя, ограничения на доступ к системным ресурсам и приложениям. Профили могут быть индивидуальными или общими. В последнем случае администратор может создать специальный профиль и присвоить его всем пользователям рабочей станции.

В таких системах обязательно присутствуют развитые средства коммуникации в сетях ЭВМ – программы просмотра и создания факсов, системы электронной почты и службы удаленного доступа и др.

Клиентские ОС обязательно содержат средства, позволяющие им поддерживать совместную работу с подобными системами других производителей. Например, глобальная служба каталогов Novell, встроенная в Windows NT Workstation, реализует интеграцию с сетями на платформе Novell NetWare, обеспечивая выполнение сценариев входа в систему NetWare и функции доступа к файлам и каталогам.

Современные серверные ОС обеспечивают следующие, нижеперечисленные режимы работы в сетях ЭВМ:

- режим файл-сервера обеспечивает коллективное использование файлов;
- все файловые ресурсы, независимо от того, на каком диске они расположены, могут быть предоставлены для совместного доступа;
- в качестве рабочих станций могут выступать компьютеры, на которых установлены клиентские операционные системы.

Режим сервера печати позволяет подключать и предоставлять в совместное пользование необходимое число принтеров. Они могут быть подключены локально или по сети с помощью протоколов TCP/IP. Чтобы пользователь, который работает на автоматизированном рабочем месте (АРМе) с клиентской ОС, мог подключиться к удаленному принтеру, предоставляемому в режиме сервера печати, ему достаточно выбрать этот принтер из списка доступных. При этом используется драйвер, установленный на сервере.

В режиме сервера приложений обеспечивается выполнение с АРМа клиента приложений под управлением сервера, таких, как:

- системы управления базами данных;
- системы информационного обмена;
- системы управления производством, документооборотом, финансами и другими приложениями.

Сервер резервирования данных предоставляет возможность резервного копирования файлов. Администратор системы определяет пользователя, ответственного за эту операцию, и только он регулярно

выполняет копирование данных на выделенный для этого носитель. Эта операция, как правило, автоматизирована.

В режиме удаленного доступа взаимодействуют две ОС – серверная, установленная на ЭВМ-сервере, и клиентская, установленная на АРМах. Пользователь АРМа, связанного с сетью через сервер удаленного доступа, чувствует себя работающим непосредственно в сети – он может осуществлять доступ к файлам и данным, печатать документы, обмениваться с коллегами сообщениями по электронной почте. Такой прозрачный доступ удобен администраторам системы и для связи территориально удаленных филиалов предприятий. Специальный набор протоколов (PPP) позволяет осуществлять удаленный доступ в условиях разнородной сети. Поддержка PPP гарантирует возможность удаленного доступа через любой стандартный PPP-сервер удаленного доступа. Современная серверная ОС должна обеспечивать доступ к сети для пользователей, применяющих средства удаленного доступа других производителей.

Режим сервера связи обычно подразумевает возможность соединения между собой различных сегментов разнородных сетей. Для прозрачного подключения к разнородным сетям создаются продукты, обеспечивающие клиентскую и серверную части не зависящим от платформы протоколом доступа к файлам в сети.

Иногда в серверной ОС выделяется режим обеспечения услуг Интернет, таких, как сервер Web, FTP и Gofer, хотя часто эти услуги реализуются самостоятельными программами, обеспечивающими распространение в Интернет потоков аудио- и видеоинформации.

Все основные серверные ОС имеют:

- пользовательский графический интерфейс;
- диспетчер заданий;
- службу администрирования;
- службы мониторинга сети и осуществления диагностики;
- службы каталогов и регистрации сетевых пользователей.

Администрирование сети – добавление новых пользователей, авторизация доступа к ресурсам сети, отслеживания административных, кадровых, структурных, функциональных и технических изменений – производится централизованно с АРМа системного администратора.

Кроме клиентских и серверных ОС широко используются современные **операционные системы автоматизированного проектирования** (ОС САПР) – часть программного обеспечения автоматизированного проектирования, предназначенная для управления проектированием. В настоящее время подавляющее большинство проектных и конструкторских работ выполняется с помощью соответствующих САПР – от архитектурного и строительного проектирования до автоматизированных систем создания программных средств самих САПРов.

Операционные системы реального времени – системы, обеспечивающие режим работы ЭВМ в реальном времени, используются

для управления технологическими процессами или автономными техническими системами, например космическими объектами.

В целом можно сказать, что ОС различного назначения занимают самое нижнее, базовое место в иерархии системных программных средств, что видно и на рис. 1.

1.3. Системное ПО разработки ПС

Развитие методов и средств разработки прикладного программного обеспечения последовательно шло в направлении максимальной независимости от программно-технических платформ, на которых приложения могут выполняться. Это обычный путь развития всех технических систем, который обеспечивается максимально возможным использованием стандартных компонент и соглашений о способах их объединения в системы. В индустрии программного обеспечения способы сопряжения компонент в программные системы называются интерфейсами.

Можно выделить три основных подхода к созданию прикладных программных систем (ПС), которые используют следующие технологии:

- языки программирования и интегрированные среды, в которых осуществляется вся технология разработки, – от подготовки исходных текстов программ и компонент до сборки готовых к использованию ПС;
- интегрированные пакеты для разработки приложений пользователями-непрограммистами;
- системы автоматизированного проектирования ПС и автоматизированных информационных систем (АИС) – CASE-системы.

Общей концепцией для всех трех подходов является концепция объектно-ориентированного метода создания ПС и АИС. Наблюдается тенденция сближения этих подходов на базе использования стандартных объектно-ориентированных методов и компонент, включаемых в современные ОС.

Первоначальные системы программирования состояли из разрозненных компонент, которые зачастую были изготовлены различными производителями. Исходные тексты программ подготавливались в текстовом редакторе одного производителя. **Трансляция программы** – преобразование программы, представленной на одном из языков программирования, в программу на другом языке и в определенном смысле равносильную первой, производилось **транслятором (компилятором)** – программой, выполняющей трансляцию программы, изготавливаемой другой фирмой. Наконец, **отладка программы** – обнаружение, локализация и устранение ошибок в программе вычислительной машины производилось в среде отладчика третьего производителя. **Отладчик** – это обслуживающая программа, облегчающая процесс отладки программ, реализует заданный режим выполнения

отладки, собирает диагностическую информацию, выдает промежуточные результаты.

Процесс производства ПС заканчивался **тестированием программы** (деятельность, направленная на поиск ошибок в программном средстве, допущенных при его проектировании и разработке). Транслятор также обычно выполняет диагностику ошибок, формирует словари идентификаторов, выдает для печати тексты программы. Тестирование в то время выполнялось по правилам, согласованным между разработчиком и заказчиком, общепризнанные стандарты отсутствовали. В результате всех этих работ на свет появлялось программное средство, которое выполняло поставленную в ТЗ задачу и работало изолированно от любых других программ, базированных на данной программно-аппаратной платформе.

Значительная часть программ создавалась для работы в режиме **интерпретации**, т.е. реализации смысла некоторого синтаксически законченного текста, представленного на конкретном языке программирования. В этом случае использовался **интерпретирующий компилятор** – компилятор, осуществляющий независимую компиляцию каждого отдельного оператора исходной программы. При интерпретации скорость выполнения программы существенно снижается, однако имеется возможность пошагового прослеживания и выполнения программы. Большое внимание в то время уделялось использованию при программировании **макроопределений** – программ, под управлением которых макрогенератор порождает макрорасширения макрокоманд. Этот подход позволял делать исходные тексты более короткими и читабельными при удачном обозначении макрокоманд.

Как уже говорилось выше, логика развития программирования от искусства к индустрии потребовала стандартизации и интеграции этапов жизненного цикла программных систем. Были созданы **интегрированные среды** – системы программ, обеспечивающие единообразие работ на всех этапах создания ПС и включающие все необходимые пользователю-программисту инструменты. Подробно интегрированные среды программирования были рассмотрены в курсе “Технология программирования”, далее мы приведем некоторые характеристики одной из наиболее развитых интегрированных сред Microsoft Developer Studio (MDS).

Интегрированная среда MDS объединяет различные средства разработки и делает их взаимно дополняющими частями единой системы создания программного обеспечения. Она позволяет одинаково легко разрабатывать приложения с использованием языков C++, Fortran, J++, а также получать доступ к технической информации. Использование различных средств контроля версий и создания проектов также значительно упрощается при использовании MDS. Программисту-разработчику нет необходимости привыкать к разнообразным системам разработки программного обеспечения с

различными интерфейсами и наборами команд. Для всех инструментов MDS предлагает единый интерфейс. Причем данная среда в процессе установки автоматически определяет присутствие на компьютере того или иного инструмента и включает его в свой состав.

Эта система рассчитана на поддержку всех аспектов разработки корпоративных информационных систем на базе архитектур клиент-сервер и интранет/Интернет и открывает возможности для разработчиков, желающих многократно использовать исходный код.

Ключевые возможности и инструменты новой системы программирования следующие:

- поддержка разработки и использования элементов Web-страниц;
- выделение цветами синтаксических конструкций HTML и других языков программирования;
- наличие редактора ресурсов объектов;
- фильтры графических форматов JPEG и GIF;
- поддержка элементов управления Visual Basic;
- генератор отчетов, обеспечивающий вывод отчетов в формате HTML;
- система хранения и управления для OLE-элементов и объектов C++.

Разработчики могут сохранять классы, ресурсы и файлы-заголовки классов в виде объектов, готовых для повторного использования, а также получают доступ к огромному рынку OLE-элементов, производимых третьими фирмами. В составе Visual C++ поставляется большое количество готовых компонентов C++ и OLE-элементов.

В MDS входит *Мастер приложений* (App Wizards), позволяющий создавать шаблоны приложений. Разработчики могут создавать собственные *Мастера приложений* для существующих проектов или изменять стандартные.

Утилита для просмотра классов без их предварительной компиляции встроена в рабочее окно проекта. ClassView позволяет разработчикам просматривать и редактировать свой исходный код как набор классов, а не работать с файлами, делая тем самым Visual C++ еще более объектно-ориентированным инструментом разработки.

Выборочная компиляция и редактирование связей сокращают время, необходимое для создания исполняемого модуля после внесения изменений в исходные файлы. При этом после редактирования исходного файла перекомпилируются только модифицированные функции, а не весь файл целиком: с помощью выборочного редактирования связей при изменении заголовочного файла перекомпилируются только те исходные файлы, которые затронуты изменениями, а не каждый файл, включающий заголовок.

C++ является наиболее мощной и универсальной системой разработки для современных прикладных систем. Компилятор поддерживает новые элементы для создания интерфейсов, диалоговые

окна, работу с протоколами TCP/IP, средства отладки под управлением ОС Windows 95 и Windows NT 4.0.

В состав среды C++ входит библиотека классов MFC, которая стала де-факто индустриальным стандартом разработки приложений для Windows. Для этой системы характерны:

- полная реализация Win32-элементов;
- возможность использования инструмента для связи с базами данных;
- создание хранимых процедур, выполняющихся на серверном компьютере;
- повышение производительности программирования благодаря использованию интегрированного в среду MDS средства групповой разработки и контроля версий;

и др.

C++ в MDS содержит набор инструментов для работы с удаленными базами данных, который позволяет:

- просматривать содержимое баз данных и таблиц на сетевых серверах непосредственно из MDS;
- редактировать хранимые процедуры и выполнять SQL-запросы;
- использовать специальный удаленный отладчик для отладки исходных текстов, находящихся на клиентской машине;
- устанавливать приложения на многих платформах;
- использовать объектные компоненты, разработанные на любом языке программирования;
- включать новые библиотеки классов;
- разрабатывать компактные приложения для распространения в Интернете/интранете и исполнения на сетевых клиентах;
- компилятор поддерживает мультимедиа-расширение MMX набора команд микропроцессоров корпорации Intel и полный набор команд Intel Pentium Pro.

Язык Java приобрел статус средства разработки приложений для Интернета. У Java, без сомнения, есть ряд достоинств. Он создан как переносимый, надежный и простой объектно-ориентированный язык с элегантными библиотеками классов. Java немало позаимствовал у C++, отказавшись в то же время от некоторых эффективных, но экстравагантных и недостаточно надежных возможностей, отдаваемых C++ в руки программистов. Java – вполне доступное пониманию средство разработки сложных сетевых решений и, уже сегодня, – один из самых популярных инструментов разработки приложений для Интернет. В пакете Visual J++ (в среде MDS) объединились элегантность Java с мощностью базовых технологий Windows. Это позволяет создавать с помощью Visual J++ большие коммерческие приложения в среде клиент-сервер и Интернет.

При помощи компилятора J++ (в среде MDS) можно максимально ускорить процесс разработки Java-приложений, использовать все

преимущества существующих и развивающихся программных стандартов. Этот инструмент позволяет:

- создавать независимый от платформы исходный текст на языке Java при минимальных затратах на редактирование, компиляцию и отладку;
- отлаживать Java-программы при помощи визуального Java-отладчика, способного работать одновременно с несколькими приложениями на Java;
- просматривать и изменять значения как переменных, так и целых выражений в процессе отладки;
- при помощи разнообразных Мастеров разрабатывать многопоточные Java-приложения, управляемые событиями;
- быстро изучить подходы к программированию на языке Java при помощи электронных справочников, примеров исходного кода, диаграмм иерархии классов и полной документации;
- создавать экранные формы, меню, инструментальные панели, значки, картинки и курсоры или повторно использовать существующие формы при помощи специального редактора ресурсов;
- использовать компонентную модель построения программ (COM) для разработки программ;
- интегрировать промышленные стандарты доступа к данным для создания приложений, активно работающих с данными.

Создана профессиональная библиотека классов для Java. Это – богатый набор написанных на Java модулей для создания сложных приложений с развитым графическим интерфейсом и доступом к базам данных. Он значительно облегчит использование Visual J++ независимыми разработчиками в качестве платформы для разработки коммерческих приложений.

Параллельно с интегрированными средами программирования в составе системного ПО развивались автоматизированные средства обработки больших массивов данных. Эти средства получили название **систем управления базами данных** (СУБД), под которыми понимается совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия ее с прикладными программами.

Появление персональных ЭВМ дало стимул для разработки офисных **интегрированных систем** в виде нескольких взаимосвязанных пакетов прикладных программ, в том числе текстовый редактор, электронная таблица, база данных, деловая графика, средства коммуникации и др.

Офисные системы предоставляют пользователю-непрограммисту эффективный способ организовать свою работу, обеспечить надежную и удобную связь с внешним миром и быстро получить необходимый результат. Как правило, они включают в себя:

- текстовый процессор;

- электронную таблицу ;
- инструмент подготовки и проведения презентаций;
- персональный информационный менеджер для управления и организации работ на АРМе;
- систему управления базами данных.

В офисные пакеты включают также средства для быстрой и эффективной справочной поддержки пользователя, которые помогают быстро найти ответы на большинство возникающих вопросов по выполнению тех или иных операций.

Текстовые процессоры офисных пакетов позволяют создавать практически любые профессионально оформленные документы. Специальные встроенные средства позволяют шаг за шагом создавать деловые документы в общепринятых международных стандартах. Могут быть автоматически установлены все параметры делового письма, его оформление, вставлены общие места (например, обратный адрес и адрес получателя), а затем все письмо будет отредактировано. Все это позволяет экономить время и сосредотачивать внимание не на форме, а на содержании документа.

В большинстве текстовых процессоров реализована фоновая проверка орфографии. По мере введения текста этот механизм проверяет наличие ошибок в словах и выделяет слова, содержащие ошибки. Простым нажатием кнопки мыши можно получить список слов, на которые можно заменить неверно введенное слово. В ряде процессоров проверяется наличие ошибок не только в словах, но и в выражениях.

Средство для работы с электронными таблицами – это простой, удобный и эффективный инструмент, позволяющий проанализировать данные и, при необходимости, проинформировать о результате заинтересованную аудиторию, используя Интернет. Это средство может распознавать наиболее распространенные ошибки, допускаемые пользователями при вводе формул в ячейку. Например, автоматически исправляются ошибки, связанные с неправильными ссылками, полученными в результате перемещения ячеек. При создании таблиц пользователи часто применяют возможности оформления для более наглядного отображения информации. Например, форматировать ячейки с использованием свойств, таких, как слияние двух и более ячеек в одну, поворот текста в ячейке на произвольный угол, начертание текста в ячейке с отступом и т. д. В электронных таблицах встроена библиотека типов представления данных в виде диаграмм. Разновидности диаграмм позволяют отображать данные более наглядно. Пользователь может строить диаграммы в виде цилиндрических столбиков, конусов, пирамид, пузырей и т. д.

Инструмент подготовки и проведения презентаций позволяет четко структурировать, иллюстрировать и профессионально представлять идеи и достижения. Такие презентации используются при неформальных

встречах, официальных презентациях для аудитории или для использования в Интернет. В качестве средства отображения можно использовать проецирование на экран через жидкокристаллическую панель, слайды, цветные и черно-белые прозрачные пленки, страницу Web-сервера.

Реляционные системы управления базами данных используются для управления любой информацией и принятия оптимальных решений. Эти СУБД могут сводить воедино данные из самых разных источников (электронные таблицы, другие базы данных), помогают быстро находить необходимую информацию, доносить ее до окружающих с помощью отчетов, графиков или таблиц, а также предлагают необходимые инструменты построения готового решения для конкретного приложения. В СУБД включены следующие компоненты:

- средство создания баз данных, которое позволяет выбрать из библиотеки, состоящей из различных типов баз данных, тот, который больше всего подходит для решения конкретной задачи;
- средство импорта-экспорта данных, которые хранятся в каком-то формате, отличном от формата СУБД, преобразует данные в нужный формат;
- анализатор таблиц позволяет быстро создать из большой “плоской” таблицы данных реляционную базу данных с несколькими таблицами и взаимосвязями между ними;
- средство создания запросов позволяет произвести выборку из имеющихся данных из одной или нескольких таблиц, при этом можно оперировать несколькими таблицами, связывая отдельные поля таблиц произвольным образом, позволяет создавать гибкие разветвленные структуры данных, удобные в управлении и эффективные в использовании.

Средство совместного использования данных позволяет разделить базу на два файла, в первый из которых помещаются собственно таблицы с данными, а во второй – запросы, формы, макросы и модули, для организации процесса обработки одного массива данных несколькими пользователями.

В современные офисные пакеты встраивают инструменты и технологии для поддержки работы в Интернете. Одной из таких функций является возможность создания в документах гипертекстовых ссылок. Они аналогичны ссылкам на страницах Web-серверов и позволяют быстро перемещаться внутри документа, переходить к другим документам в любом формате, а также к любым ресурсам в сетях Интернет/интранет. Например, имеется документ, содержащий отчет о финансовых показателях за календарный период. Отчет может быть иллюстрирован диаграммами и таблицами. Бывают ситуации, когда внутри документа необходимо сослаться на другой документ – таблицу, содержащую большой объем данных. Нет необходимости непосредственно вставлять ее в текстовый файл. В этом случае в текст можно поместить ссылку на

таблицу с данными. Щелкнув кнопкой мыши на этой ссылке, можно получить документ, загруженный в приложение, в котором он был создан. Если ссылка указывает на HTML-страницу, то она будет автоматически открыта в браузере.

Гипертекстовые ссылки могут быть созданы в любом приложении и позволяют сослаться на любой документ, ресурс в интранет-сети или просто на каталог в корпоративной сети. Ссылка может быть относительной (т.е. указывать смещение относительно текущего каталога) или абсолютной (т.е. указывать полный путь к документу).

Все компоненты пакета, как правило, используют единую панель инструментов для работы с ресурсами сетей Интернет-интранет. Эта панель облегчает работу с HTML-страницами, просмотр ресурсов Интернет и интранет-сетей. Кнопки панели полностью идентичны кнопкам, имеющимся на панели инструментов соответствующих браузеров.

В офисные пакеты включают средства, которые позволяют создавать страницы в формате HTML, готовые для публикации на Web-сервере. Знания о языке HTML при этом не требуются. Достаточно просто сохранить готовый документ как HTML-страницу.

При создании таких страниц можно использовать стандартные атрибуты Web-страниц, в частности:

- можно добавлять фоновый рисунок, который будет присутствовать при просмотре Web-страницы, выбрать готовую текстуру из библиотеки или создать свою собственную;
- можно вставлять различные готовые элементы оформления, в т.ч. горизонтальные линии, рисунки, видеоролики, звуки, мигающий текст и др.

Средства электронных таблиц содержат встроенные модули, позволяющие помещать на Web-сервер документы, созданные в среде процессора электронных таблиц. К их числу относятся средства сохранения документа в формате HTML и средства просмотра содержимого документов для пользователей, не работающих с процессором электронных таблиц. Имеется возможность импортировать данные из HTML-документов, найденных на Web-сервере, восстанавливая при этом формат и оформление таблицы. После импорта данные доступны для выполнения любых операций.

1.4. Системное ПО разработки АИС

Прикладные программные системы являются компонентами автоматизированных информационных систем, в том числе **управленческих информационных систем**, основной функцией которых является обеспечение руководства информацией. Первая концепция АИС, широко распространенная в 60-х и начале 70-х гг. заключалась в том, что системотехники определяли информационные потребности

отдельных менеджеров в какой-либо организации и проектировали системы для обеспечения этой информацией постоянно или по запросам. Новый класс АИС образуют системы поддержки принятия решений, которые дают менеджерам еще большую свободу в использовании информации, предоставляемой машиной. АИС этого класса представляют собой симбиоз учрежденческих информационных систем (в том числе, персональных вычислительных средств менеджеров, на которых они работают сами, без традиционных стандартных баз данных) и средств обработки информации. Системы поддержки принятия решений (СППР) предназначены для того, чтобы менеджеры могли создавать и использовать собственные базы, равно как и работать с внутрифирменными базами данных, формируя собственные информационные запросы без помощи специалистов по разработке систем.

Современные АИС разрабатываются с помощью **информационных систем автоматизированного проектирования**, которые представляют собой системы, имеющие альтернативное программное обеспечение и операционную систему автоматизированного проектирования, позволяющую выбирать совокупность машинных программ применительно к заданному объекту проектирования или классу объектов проектирования.

К середине 1990-х гг. в области информационных технологий четко обозначились следующие факторы:

- развитие электроники значительно увеличило ресурсы ЭВМ, которые можно было использовать для автоматизации проектирования, с тем, чтобы уменьшить его трудоемкость и длительность;
- сформировался объектно-ориентированный подход, хорошо структурирующий задачу как таковую, а также и ее решение в виде прикладной системы;
- на базе объектно-ориентированного подхода стали развиваться визуальные средства быстрой разработки, основанные на компонентной архитектуре;
- стандартизировались основные методы работы с базами данных;
- все большее количество АИС стало принимать распределенный характер благодаря развитию телекоммуникационных сетей и концепции открытых систем.

Эти и другие факторы явились предпосылками для реальных изменений объектов проектирования – программных средств и баз данных АИС, и в частности возможность их переноса на различные операционные и аппаратные платформы. Все большее значение и массовость приобретают разработки сложных АИС, возросли объемы их программного и информационного обеспечения и требования к их качеству и надежности. В то же время, происходит накопление проверенных на практике высококачественных компонент, которые можно многократно использовать в различных версиях программного обеспечения АИС и на различных платформах.

Основная цель развития современных технологий проектирования АИС состоит в повышении экономической эффективности всего жизненного цикла в различных проблемно-ориентированных областях. Это достигается снижением трудоемкости, ускорением и упрощением проектирования всей совокупности компонент, проведением комплексной автоматизации технологий обеспечения всего жизненного цикла АИС.

Совокупность методов и инструментальных средств автоматизации технологического процесса разработки сложных АИС объединяется под названием CASE (Computer Aided Software Engineering – автоматизированное проектирование программных средств).

CASE-технология регламентирует порядок организации и проведения работ, неавтоматизированного и автоматизированного выполнения технологических операций, направленных на получение в имеющихся организационно-технических условиях готовой АИС с заданными функциями и качеством. Для достижения этой цели разработан и активно используется ряд принципов:

- максимально возможное повышение уровня абстракции при описании компонент и действий над ними на различных этапах проектирования;
- сокрытие всей информации, избыточной для данного этапа или объекта проектирования;
- модульность и иерархия в структурном построении программных и информационных компонент;
- унификация правил проектирования, структурного построения и взаимодействия компонент между собой и с внешней средой;
- поэтапный контроль полноты и качества решения функциональных задач.

Интегрированные CASE-средства служат для извлечения знаний из заказчика на этапе проведения обследования, а также для проектирования концептуальной и логической структур баз данных прикладной системы. Кроме того, с их помощью осуществляется сопровождение и развитие проекта. Основными функциями CASE-средств являются:

- объектно-ориентированное системное и логическое проектирование программных средств и баз данных;
- планирование и оценка затрат ресурсов на разработку программных средств и баз данных;
- стратегическое планирование и управление проектами на всем жизненном цикле;
- анализ требований, структурное проектирование ПС и БД, разработка и применение спецификаций требований;
- организация и управление базами данных и хранилищами проектов;
- повторное использование отработанных программных компонент, а также перенос их на иные операционные и аппаратные платформы.

Языки четвертого поколения (4GL) являются средством разработки приложений. Они легки в изучении и намного уменьшают размер кода, требуемого программистам для проектирования, записи и тестирования приложений. В результате достигаются значительно меньшие программистские затраты и возрастает производительность по сравнению с языками низкого уровня. 4GL также облегчают программистам следующие задачи:

- поддержку и модификацию программных приложений;
- написание более строгой программной документации;
- осуществление быстрого макетирования приложений;
- управление проектированием программных средств.

Многие 4GL предоставляют возможность лицам, не являющимся профессиональными программистами, решать самостоятельно некоторые задачи, например написание запросов и отчетов БД. На уровне 4GL сообщается приложению, что следует сделать, 4GL автоматически решает, как это сделать.

Под термином 4GL подразумеваются не только конкретные языки программирования, но и множество средств для разработки программного обеспечения, поставляемое вместе с языками. К этим средствам относятся:

- отладчики;
- редакторы текстов и баз данных;
- генераторы приложений и меню;
- средства создания форм и отчетов;
- средства форматирования экрана;
- системы презентации.

1.5. Интерфейсы и оболочки

По мере роста мощности ЭВМ увеличиваются затраты на диалоговую компоненту программного обеспечения. Термин «эффективность» постепенно изменил свое значение. Если раньше он отражал такие характеристики, как процессорное время и объем занимаемой памяти, то теперь под ним понимают простоту разработки, легкость сопровождения и удобство работы с программой. Поэтому затраты на исследование и разработку пользовательского интерфейса являются оправданными. **Интерфейс** – это совокупность средств и правил, обеспечивающих взаимодействие устройств цифровой вычислительной системы и (или) программ, а **интерфейс пользователя** – программные и аппаратные средства взаимодействия оператора или пользователя с программой или ЭВМ.

Большинство современных пользовательских интерфейсов основываются на аналогичных идеях:

- активное использование «мыши»;
- ориентированность на объекты;

- графика и имитация процессов и явлений;
- возможность использования алгоритмов, знакомых каждому человеку из его обыденной жизни.

Разнообразие аппаратных и системных платформ, на которых должно будет работать это программное обеспечение, требует его переносимости на уровне исходного кода. Вышеизложенные требования логически приводят к идее переносимого унифицированного программного инструмента для создания пользовательских интерфейсов, точнее, программной системы (модуля, блока), которая обеспечивает интерфейс с пользователем.

Рассмотрим пример современной стандартной системы для построения интерфейсов (СПИ) – это система для создания *графического пользовательского интерфейса*. Она построена по схеме клиент-сервер. Взаимодействие клиента и сервера происходит в рамках соответствующего протокола прикладного уровня. В качестве транспортного протокола может служить как локальный, так и сетевой. Это означает, что различия в архитектуре клиента и сервера не играют никакой роли и их совместимость обеспечивается стандартом протокола.

Система обеспечивает графический вывод на экран, воспринимает сигналы от устройств ввода, таких, как клавиатура и мышь, и передаёт их программам. Устройство вывода может иметь более одного экрана и обеспечивается вывод на любой из них.

Благодаря своей архитектуре, СПИ свободно используется в распределенных вычислительных системах, например в сетях TCP/IP (Internet), и позволяет пользователю (за дисплеем) общаться со многими программами одновременно. Чтобы вывод из них не смешивался, система создаёт на экране дисплея «виртуальные подэкраны» – окна. Каждое приложение (как правило) рисует только в своём окне (или своих окнах). СПИ предоставляет набор средств для создания окон, их перемещения по экрану, изменения их размеров, вывода в них и т.п.

Как правило, программы имеют набор конфигурационных параметров – ресурсов. Это может быть цвет окна, различные параметры текстового шрифта и многое другое. Система стандартизует способ задания ресурсов приложений, управления ими и содержит ряд процедур для работы с ними. Эта совокупность функций называется менеджером ресурсов.

СПИ функционирует согласно идеологии управляемости событиями – она организует общение между самими программами и между программами и внешней средой посредством событий. Событие есть единица информации, идентифицирующая происходящие в системе изменения или действия. По идентификатору события можно получить информацию о нём: вид события, его характеристики, где оно произошло и т.п.

Система представляет собой совокупность программ и библиотек. Ядром её является специальная программа – сервер. Именно сервер

знает особенности конкретной аппаратуры, знает, что надо предпринять, чтобы вывести какой-либо графический объект на экран. Он же умеет воспринимать и обрабатывать сигналы, приходящие от клавиатуры и мыши. Сервер общается с программами-клиентами, посылая им или принимая от них пакеты данных.

Если сервер и клиент находятся на разных машинах, то данные посылаются по сети, а если на одной – то используется внутренний канал. Например, если сервер обнаруживает, что пользователь нажал кнопку мыши, то он подготавливает соответствующий пакет (событие) и посылает его тому клиенту, в чьём окне находился курсор мыши в момент нажатия кнопки. И наоборот, если программе надо вывести что-либо на экран дисплея, она создаёт необходимый пакет данных и посылает его серверу. Описание этого взаимодействия, форматов пакетов и т.п. и составляет спецификацию протокола.

Чтобы программировать, совсем необязательно знать детали реализации сервера и протокола. Система предоставляет стандартную библиотеку процедур, с помощью которых программы осуществляют доступ к услугам СПИ на высоком уровне. Так, для того чтобы вывести на экран точку, достаточно вызвать соответствующую стандартную процедуру, передав ей требуемые параметры. Эта процедура выполнит всю работу по формированию пакетов данных и передаче их серверу.

Окно – это базовое понятие в СПИ. Оно представляет, обычно, прямоугольную область на экране, предоставляемую системой программе-клиенту. Клиент использует окно для вывода графической информации. Окно имеет внутренность и край. Основными атрибутами окна являются ширина и высота внутренности, а также ширина (толщина) края. Эти параметры называются геометрией окна.

С каждым окном связывается система координат, начало которой находится в левом верхнем углу окна. Ось x направлена вправо, а ось y – вниз. Единица измерения по обеим осям – пиксел. СПИ позволяет программе создавать несколько окон одновременно. Они связаны в иерархию, в которой одни являются «родителями», а другие – «потомками». Сам сервер на каждом экране создаёт одно основное окно, являющееся самым верхним «родителем» всех остальных окон. Это окно называется корневым.

Окна могут располагаться на экране произвольным образом, перекрывая друг друга. Имеется набор средств, пользуясь которыми, программа-клиент может изменять размеры окон и их положение на экране. Чтобы управлять окнами с помощью мышки или клавиатуры, имеется менеджер окон. Однако менеджер не может корректно управлять окнами, ничего о них не зная. Клиенты могут сообщать менеджеру свои пожелания относительно окон двумя способами:

- при создании окна могут быть переданы «рекомендации» о начальном положении окна, его геометрии, минимальных и максимальных размерах и т.д.;

- можно использовать встроенный в СПИ способ общения между программами – механизм свойств.

Система предназначена для работы на растровых дисплеях. Число бит на пиксел называют *глубиной* или *толщиной* дисплея. Биты с одинаковыми номерами (одинаковые двоичные разряды) во всех пикселах образуют как бы плоскость, параллельную экрану. Её называют *цветовой плоскостью*. СПИ позволяет рисовать в любой цветовой плоскости, не затрагивая остальные. Значение пиксела не задаёт цвет точки на экране непосредственно, но задаёт номер ячейки в специальном массиве, в которой и хранится значение цвета, т.е. значение пиксела задаёт номер цвета в текущей палитре. СПИ имеет большой набор процедур, позволяющих рисовать графические примитивы: точки, линии, дуги, текст; работать с областями произвольной формы.

В системе встроены средства для обмена информацией между программами-клиентами. Для этого используется механизм свойств. Свойство – это информационная структура, связанная с некоторым объектом, например окном, доступная всем клиентам. Каждое свойство имеет имя и уникальный идентификатор. Обычно, имена свойств записываются большими буквами. Идентификаторы используются для доступа к содержимому свойств с тем, чтобы уменьшить объём информации, пересылаемой между клиентами и сервером. Предусмотрен ряд процедур, позволяющих перевести имя свойства в уникальный идентификатор, и, наоборот, по идентификатору получить необходимые данные. Некоторые свойства и соответствующие им идентификаторы являются предопределёнными в СПИ.

В настоящее время определены обобщенные требования к системе управления пользовательского интерфейса (СУПИ) – как составной части СПИ.

Выделяют три объекта, для каждого из которых ставятся различные цели при разработке СПИ.

1. Интерфейс с пользователем:
 - согласованность;
 - поддержка пользователя разного уровня;
 - обеспечение обработки ошибок и восстановления.
2. Разработчик программного обеспечения:
 - предоставление абстрактного языка для конструирования интерфейса пользователя;
 - предоставление согласованных интерфейсов для связанных прикладных задач;
 - обеспечение простоты изменения интерфейса на стадии его проектирования (быстрое создание прототипа);
 - упрощение разработки повторным использованием программных компонент;
 - обеспечение простоты изучения и использования прикладных программ.

3. Конечный пользователь:

- согласованность интерфейса по прикладным программам;
- многоуровневая поддержка сопровождения или функций помощи;
- поддержка процесса обучения;
- поддержка расширяемости прикладных программ.

Эти цели определяют следующие функциональные характеристики

СПИ:

- работа с входными устройствами;
- проверка допустимости ввода;
- обработка ошибок пользователя;
- реализация обратной связи;
- поддержка обновления/изменения данных прикладной задачи;
- поддержка задач развития интерфейса;
- синтаксическая поддержка.

Наиболее часто используется модель, в соответствии с которой

СПИ состоит из трёх компонент:

- системы представления, обеспечивающей низкоуровневый ввод и вывод;
- системы управления диалогом, обрабатывающей лексические единицы, получаемые в системе представления, в соответствии с синтаксисом диалога;
- модели интерфейса прикладной программы, представляющей семантику диалога и управляющей функциональностью прикладной программы.

Конкретные реализации моделей основываются на различных способах спецификации интерфейса, среди которых можно выделить следующие типы:

- языковая;
- графическая;
- автогенерация по спецификации прикладной задачи;
- объектно-ориентированный подход.

Каждая из этих спецификаций имеет свои особенности. Для языковой спецификации применяется специальный язык, чтобы задавать синтаксис интерфейса. Такими языками могут служить:

- сети меню;
- диаграммы состояний и переходов;
- контекстно-свободные грамматики;
- языки событий;
- декларативные языки;
- обычные языки программирования;
- объектно-ориентированные языки.

Этот подход приложим к широкому кругу прикладных задач. Его недостатком является то, что разработчик диалога должен обладать профессиональной программистской подготовкой.

Графическая спецификация связана с определением интерфейса с помощью размещения объектов на экране (визуальное программирование, программирование по примерам). Она проста для использования непрограммистами, но трудна в реализации; поддерживает лишь ограниченные интерфейсы. Здесь также существуют разные формы реализации:

- размещение на экране интерактивных средств (меню, кнопки и т.п.) и их привязка к фрагментам, написанным разработчиком интерфейса;
- сеть статичных страниц (кадров), содержащих тексты, графики, интерактивные средства; спецификация по демонстрации.

В методе *автогенерации по спецификации прикладной задачи* интерфейс создаётся (точнее, делается попытка создать) автоматически по спецификации семантики прикладных задач. Однако ввиду сложности адекватного описания интерфейса трудно ожидать скорого появления систем, реализующих такой подход в полной мере. Возможные реализации создания интерфейса:

- на основе списка процедур прикладной программы;
- по типам параметров процедур;
- на основе определения семантики прикладной задачи, описываемой на специальном языке.

Объектно-ориентированный подход связан с принципом, при котором пользователь взаимодействует с индивидуальными объектами, а не со всей системой как единым целым.

В заключение приведём в качестве примера описание функций СПИ одной из фирм, созданной на базе требований, приведенных в п. 1.5.

Проектирование интерфейса – пользователь создает интерфейс в интерактивном режиме, используя предопределённые элементы – заготовки.

Спецификация ресурсов – реализует простую установку параметров для заготовок. Многообразие ресурсов заготовок и их взаимодействия делает задачу установки параметров чрезвычайно сложной, поэтому везде, где это возможно, предоставляется предопределённая установка соответствующего выбора.

Спецификация поведения интерфейса – описывается на Си-подобном командном языке.

Связь между интерфейсом и прикладной задачей – реализована вызовом функции прикладной задачи из описания или с помощью разделяемых переменных (активных значений), разделяемые переменные могут быть любого типа.

Тестирование интерфейса и его поведения (режим попытки) – в этом режиме интерпретируется описание, связанное с какими-либо событиями, но без вызова функций прикладной задачи.

Эффективность конечного приложения – результат проектирования реализуется одним из двух способов:

- интерпретацией описания;
- компиляцией интерфейса вместе со всеми описаниями в Си-код.

1.6. Системное ПО телекоммуникационных сетей

В п. 1.2.2 были рассмотрены сетевые ОС, но состав системного сетевого ПО не исчерпывается этим классом программ. Ведь под **телеобработкой данных** понимается совокупность методов, обеспечивающих пользователям дистанционный доступ к ресурсам систем обработки данных и ресурсам средств связи. Если **телекоммуникационный метод доступа** обеспечивает установление связи и передачу данных между ЭВМ и удаленными или локальными терминалами и абонентскими пунктами, то хранение и обработку данных обеспечивают системы управления распределенными базами данных.

Сетевые программные среды. В частности, при разработке информационных систем используют следующие средства:

gorher – для доступа в диалоговом режиме с помощью иерархического меню к информационным файлам;

archie – для поиска ключевых строк в файлах меню gorher серверов;

WWW – для организации гипертекстовых информационных систем для работы в on-line режиме с локальными и удаленными пользователями;

WAISE – для поиска в распределенных и удаленных базах данных используется естественный язык запросов.

Перечисленные средства используют идеологию “клиент-сервер” и имеют реализацию для различных платформ, в том числе и в форме свободно распространяемого программного обеспечения.

Широко распространяются также системы управления распределенными базами данных (СУРБД), которые создают у пользователя иллюзию локальности данных, хранимых на самом деле в различных местах одной сети (локальной или глобальной) или даже в различных сетях. В настоящее время фактически все ведущие фирмы-разработчики СУБД заявляют о поддержке распределенных данных и ведут работы по созданию новых и дальнейшему усовершенствованию уже имеющихся систем.

Принято считать, что продукт, претендующий на статус СУРБД, должен поддерживать следующий конкретный список возможностей:

1) *Распределенный словарь-справочник данных.* Здесь записываются тип и место хранения данных, а также способ доступа к ним.

2) *Прозрачная двухфазная фиксация.* Протокол двухфазной фиксации обеспечивает согласованность данных. Если в транзакции участвует более одного узла, этот протокол позволит ей завершиться только в том случае, когда каждый узел выполнит требуемые задачи. Если хотя бы один узел не может выполнить свою задачу, вся транзакция отменяется.

3) *Горизонтальная и вертикальная фрагментация.* Таблица базы данных разбивается на строки или столбцы, которые можно перемещать между узлами.

4) *Независимость копирования.* Пользователи могут копировать данные из другого узла, не оказывая влияния на производительность прикладной программы и согласованность данных.

5) *Многоузловые таблицы.* Таблицы данных на одном узле могут быть объединены с таблицами данных на других узлах.

6) *Оптимизация распределенных запросов.* Обеспечиваются наилучшие методы выполнения сложных функций, таких, как объединение таблиц в территориальных сетях. Определяется сетевой канал, способный справиться с объемом передаваемой информации, и узел, имеющий достаточную вычислительную мощность для объединения таблиц.

7) *Ограничение многоузловой целостности.* Поддерживается ссылочная целостность данных – при обновлении записи в главной БД связанные с ней записи в отдаленных узлах автоматически обновляются.

8) *Локальная автономия.* Администраторами БД гарантируется контроль над данными, хранящимися на их узле.

9) *Непрерывное функционирование.* Обеспечивается постоянная работа в локальном узле вне зависимости от функционирования других узлов.

10) *Независимость от местоположения.* Передача данных из одного узла в другой не оказывает влияния на работу прикладной программы.

11) *Управление распределенными транзакциями.* Гарантируется восстановление базы данных, распределенной по нескольким узлам, в соответствии с существующими ограничениями целостности.

12) *Глобальная параллельная обработка и восстановление после взаимной блокировки.* Осуществляется управление блокировками записей данных в распределенной БД, определяется и разрешается любая ситуация, в которой два узла взаимно блокируются при попытке доступа к данным друг друга.

13) *Независимость от аппаратных, программных, сетевых средств и от СУБД других производителей,* обеспечиваемая с помощью встроенной поддержки или шлюзов. Распределенная СУБД встраивается в существующие среды и системы данных.

По мнению экспертов, наиболее полно удовлетворяет этим требованиям на данный момент СУБД Ingres. Эта СУБД поддерживает UNIX-платформы, Windows, OS/2, Macintosh и используется во многих сетях, предоставляя доступ к другим СУБД. СУБД Ingres предоставляет функции ведения глобального словаря, извлекая информацию из всех локальных словарей, и осуществляет оптимизацию запросов. Недостатком сетевой обработки является и механизм ведения журнала транзакций, при котором журнал не может расширяться при необходимости. Это приводит к необходимости резервирования больших объемов дискового

пространства, что снижает быстродействие системы. Достоинством системы является сильный механизм оптимизации запросов.

Все производители СУБД в настоящее время поддерживают стандарт ANSI SQL-89 и ряд собственных расширений. Кроме того:

- обеспечивается прозрачный доступ к данным для конечного пользователя, т.е. возможность запрашивать, обновлять и рассматривать данные во многих узлах;
- предусмотрены глобальная и локальная блокировки БД на различных уровнях: файл, таблица, запись;
- всеми производителями поддерживается целостность БД.

Рынок распределенных СУБД пока еще находится в стадии становления. Доступные на сегодня распределенные СУБД по производительности в целом не дотягивают до стандартов, требуемых для важных приложений. При организации запросов и управлении блоками в распределенной среде по территориальной сети проходит очень много сообщений, что замедляет отклик системы. Большинство систем обеспечивают адекватную производительность в высокоскоростных локальных сетях и значительно хуже работают в распределенной сети с низкоскоростными линиями.

На данном этапе альтернативой распределенным СУБД служат системы на базе архитектуры “клиент-сервер” (как было сказано выше), когда данные хранятся в сервере централизованной базы данных SQL и пользователи в локальной сети разделяют доступ к этой общей БД.

2. ОСНОВЫ ОПЕРАЦИОННЫХ СИСТЕМ

2.1. Развитие, назначение и структура операционных систем

2.1.1. Основные понятия

Первые упоминания об операционной системе как самостоятельной программе со специальными функциями появились в 1953 г. В одном из университетов была создана программа для обслуживания учебных приложений. Ее целью было обеспечение безостановочного прохождения через ЭВМ последовательности вычислительных работ и предоставление возможности пользоваться хранимой на ЭВМ библиотекой служебных программ. Первые ОС были ориентированы на *совмещение операций* по запуску программы и ее выполнения при последовательном расположении в “пакете” ряда независимых программ. Существенной чертой пакетной обработки являлось то, что *каждая задача имела в своем распоряжении все ресурсы ЭВМ*, исключая лишь резидентную часть памяти, в которой постоянно находилось **ядро операционной системы** – резидентная часть операционной системы, управляющая процессами операционной системы и распределяющая для них физические ресурсы.

В последствие были предприняты попытки теоретического осмысления места операционных систем как особого вида программного обеспечения. Они сводились к следующему.

Во время выполнения программы различают три отдельных объекта:

- последовательность команд, или *процедура*, которая определяет программу;
- процессор, который выполняет процедуру;
- среда – часть окружающего мира, которую процессор может непосредственно воспринимать или изменять.

Программа рассматривается как выполнение последовательности шагов. Пусть эти шаги будут элементарными, отвлекаясь от состояния различных частей машины во время действительного выполнения каждого шага. Это означает, что “среда” должна включать любую часть машины или ее окружение, которые могут восприниматься или изменяться программой (к среде не относятся такие компоненты, как регистры адресов памяти или буфера арифметического устройства). Арифметическое устройство и управляющие регистры машины относятся к среде, поскольку они могут и восприниматься, и изменяться программой.

Тогда можно назвать следующие существенные свойства программы:

- операции, заданные процедурой, выполняются строго последовательно, каждый шаг полностью завершается, прежде чем начинается следующий;
- среда полностью находится под управлением программы и изменяется только в результате шагов, выполненных программой;
- результаты программы должны быть получены за разумный промежуток времени, но фактическое время выполнения каждой операции не имеет отношения к выполнению программы.

Несущественно также, имеется ли временной интервал между любыми двумя операциями. Наконец, совершенно не имеет значения, проходит ли выполнение программы целиком на одном процессоре. В любое время можно передавать выполнение программы другому процессору, если только этот переход не изменяет среду программы.

Все вышесказанное означает, что программа — это “закрытая система”. Во время выполнения она не подвергается внешнему воздействию и будет выполняться независимо от событий, происходящих вне ее среды, пока не остановится по собственной логике.

Очень важным свойством таких программ является возможность точного повторения их работы (если только они не обращаются к часам реального времени или к переключателям), что облегчает отыскивание и исправление ошибок.

Такие программы отличаются от операционных систем по двум основным причинам:

операционная система эффективно использует все компоненты вычислительной машины, т.е. операции различных компонент должны совмещаться;

операционная система должна отвечать на запросы в течение определенного промежутка времени.

Время ответа может быть различным – от нескольких секунд при работе в режиме диалога до нескольких миллисекунд при регистрации данных или при управлении процессом. Поскольку момент поступления этих запросов непредсказуем и не управляем системой, нельзя сказать, что последовательность операций определяется только самой системой.

Первые попытки создать программное обеспечение, которое могло бы управлять совмещенными операциями и быстро реагировать на непредсказуемые запросы, делались без всякой теоретической основы. Для простых систем результаты оказались приемлемыми, но когда среда стала более сложной, создание программного обеспечения, отвечающего требованиям эффективности и надежности, часто затягивалось на несколько лет. Главная трудность, с которой столкнулись разработчики систем, состояла в невозможности повторить условия, приведшие к появлению ошибки, и проследить причину ее появления.

Эти трудности привели к введению понятия “процесса” – одного из основных понятий, используемых при разработке операционных систем.

Процесс во многом подобен однократному выполнению программы. Он состоит из последовательности шагов, каждый из которых воспринимает или изменяет среду некоторым определенным образом. Обычно он управляется процедурой, которая сама является частью среды (хранится в ней). Процесс может быть безопасно прерван между шагами, и во время его выполнения может происходить смена процессора.

Существенная разница между процессом и выполнением программы состоит в том, что *процесс не является закрытой системой*, он может взаимодействовать с другими процессами либо явным образом, либо неявно, изменяя или воспринимая часть среды, которую он разделяет с другими процессами. Среда, разделяемая таким образом, часто называется “глобальной”, в то время как индивидуальная среда процесса называется “локальной”.

В больших вычислительных системах многочисленные процессы могут действовать одновременно. Всякая попытка понять природу системы, рассматривая действие центрального процессора, обречена на провал, так как процессор будет разделять свои ресурсы среди нескольких различных процессов. Напротив, если мы будем считать *процесс* единицей действия и не будем уделять слишком большого внимания процессору, выполняющему это действие, станет гораздо легче понять работу системы.

Как было сказано выше, процесс есть *однократное выполнение* процедуры. Это значит, что каждый процесс существует во времени. Есть момент, когда он “рождается”, и есть другой момент, когда он “умирает”, завершив свою работу. Вообще говоря, процессы инициируются другими

процессами, но есть один “главный процесс”, с которого начинается цепочка процессов. Этот главный процесс должен начинаться, когда включается вычислительная машина.

В любой момент в течение жизни процесса его продвижение может быть описано его “состоянием”. Перед началом процесса и тогда, когда он должен быть по какой-то причине прерван, переменные, описывающие его состояние, обычно объединяются в “вектор состояния”, который хранится до тех пор, пока процесс не начнется (или возобновится), и содержит информацию, достаточную для того, чтобы процесс мог продолжаться, не заметив, что он был прерван. Вектор состояния будет также содержать информацию, достаточную для формирования или восстановления локальной среды (включая адрес следующей команды). В качестве примера процесса можно привести *процесс управления файлами* – процесс, поддерживающий работу программ с внешней памятью и обеспечивающий поиск данных, контроль, обслуживание и обновление файлов.

Явная связь между процессами осуществляется при помощи средства, которое, хотя и относится к программному обеспечению, само не является процессом – это операционная система.

В последующем в этой области выделились следующие основные понятия:

ядро программного обеспечения – часть программного обеспечения, которая располагается постоянно в оперативной памяти ЭВМ;

системный ресурс – любое средство вычислительной системы, которое может быть выделено процессу;

системный процесс – часть операционной системы, выполняемая как отдельный процесс;

системный загрузчик – компонента управляющей программы, осуществляющая начальную загрузку ядра операционной системы и его инициализацию;

системная программа – программа, входящая в состав операционной системы и выполняющая управляющие и обслуживающие функции;

системная библиотека – библиотечный набор данных, содержащий определенные компоненты операционной системы;

системный журнал – набор данных, в котором регистрируется информация;

системное планирование – упорядочивание последовательности работ в системе, обеспечивающее ее максимальную эффективность;

системное имя – уникальное имя, идентифицирующее отдельное устройство вычислительной системы в данной операционной системе.

Параллельно разрабатывались операционные системы для специфического использования:

системы реального времени – системы, осуществляющие информационный обмен с другими системами, периферийными устройствами или датчиками при таких временных характеристиках, которые позволяют немедленно обрабатывать всю поступающую информацию и информацию для вывода;

системы с разделением времени – вычислительные системы, имеющие программные и технические средства реализации режима разделения времени центрального процессора между многими задачами таким образом, что выполнение отдельных заданий осуществляется в темпе их поступления.

2.1.2. Операционная система OS/360

Огромное значение для формирования теоретических и инженерных представлений о составе, структуре и логике функционирования операционных систем имел опыт разработки и использования OS/360. Эта ОС разрабатывалась в середине 60-х гг. крупнейшим мировым производителем ЭВМ фирмой IBM для ряда совместимых моделей IBM/360, которая охватывала широкий спектр применений и предлагала исключительный (для своего времени) по разнообразию набор внешних устройств. Основные черты новизны OS/360, операционной системы для IBM/360, относятся не к функциональным характеристикам, а к универсальности ее применения.

Если говорить конкретно, то OS/360 – это библиотека программ. Однако в абстрактном смысле с термином OS/360 связывают единый четкий ответ на сложную совокупность потребностей. При проектировании OS/360 была создана единая система понятий и правил, введена модульность. OS/360 предназначалась для конфигураций IBM/360, имеющих стандартный набор команд и тридцать две тысячи или более байтов (32K) основной памяти (не забывайте – это 60-е гг.).

Выше уже говорилось, что первые ОС были ориентированы на пакетную обработку. Существенной чертой пакетной обработки является то, что каждая задача имеет в своем распоряжении почти всю машину, не затрагивая лишь резидентную часть системы, постоянно присутствующую в основной памяти. В тот же период времени было развито несколько больших систем другого типа, к которым предъявлялось требование полного подчинения машинных ресурсов потребностям одной работы, идущей в “реальном времени”. Например, одна из наиболее ранних операционных систем была предназначена для проверки работы частей другой системы. Вообще говоря, эти системы реального времени либо с точки зрения применения, либо из-за структуры системы, были мало похожи на операционные системы первого (пакетного) поколения.

Основная структура OS/360 дает равные возможности и для пакетной обработки и для работы в реальном времени и ее можно рассматривать как один из первых примеров операционной системы

“второго поколения”. Целью такой системы является приспособляемость к условиям различных применений и способов работы. Другие цели, важность которых нельзя отрицать, не являлись новыми – они, в разной степени, были определены в предшествующих системах. Основными среди этих целей являются:

- увеличение производительности системы;
- малое время ответа;
- увеличение производительности программиста;
- адаптируемость программ к изменяющимся ресурсам;
- возможность расширения (открытость).

В OS/360 высокий уровень машинной производительности обеспечивался тремя способами:

- при управлении потоком работ система помогала оператору в выполнении подготовительных действий, в то время как ранее воспринятые задания исполняются;
- система позволяла задачам, относящимся к различным заданиям, параллельно использовать ресурсы в мультипрограммном режиме, что исключает необоснованные простои ресурсов;
- было уделено большое внимание разнообразию программных языков, средств отладки и удобству ввода.

Время ответа – это время, прошедшее от поступления запроса до завершения запрошенного действия. При пакетной обработке время ответа являлось относительно большим. Пользователь отдавал колоду перфокарт в вычислительный центр и через некоторое время получал напечатанные результаты. Если рассматривать всевозможные режимы прохождения работ, то имеется целый спектр времен ответа. Время прохождения работы при пакетной обработке является “красным” концом этого спектра, в то время как ответ в реальном времени находится на его “фиолетовом” конце. С учетом разнообразия требований в зависимости от применений и времен ответа OS/360 была спроектирована так, что она обеспечивала целый спектр времен ответа путем выбора соответствующих возможностей управляющей программы и варьирования приоритетов.

С точки зрения производительности труда программиста и удобства его работы в OS/360, ставилась цель достичь нового уровня гибкости путем включения в систему относительно большого количества языков программирования. Этим же целям служили макрокоманды в языке ассемблера, а также компактный язык управления заданиями, предназначенный для оформления представляемых на машину работ.

Операционная система второго поколения должна была уметь приспособляться к разнообразию условий и областей применения. IBM/360 давала возможность получить почти неограниченный диапазон машинных конфигураций и различных применений. Следовательно, операционная система была рассчитана на исключительное разнообразие применений и конфигураций. Все это приводило к необходимости иметь модульную и гибкую систему.

Целям адаптируемости служила также возможность в OS/360 строить программы, в значительной степени, независимо от конкретных устройств ввода-вывода. Создавая такие программы, можно было уменьшить или упростить проблемы, связанные с заменой устройств ввода-вывода.

OS/360 была спроектирована “открытой” для расширения – ее можно было использовать с новой аппаратурой, для новых областей применения и с новыми программами по мере того, как они появляются. Систему можно было легко приспособить к имеющимся в данный момент соглашениям и наборам символов, скомпоновать так, что связь с операторами и программистами могла осуществляться не только на английском языке, но также и на других языках. Всякий раз, когда это диктовалось изменившимися обстоятельствами, могли быть расширены функциональные возможности самой операционной системы.

Если рассматривать вычислительную систему как обобщенную машину, то можно считать, что она, подобно луковице, состоит из нескольких слоев. Немногие программисты имеют дело с самым внутренним слоем, в котором имеется только то, что обеспечивается самой аппаратурой. Например, человек, программирующий на ФОРТРАНЕ, имеет дело с внешним слоем, определяемым языком ФОРТРАН. Можно сказать, что он действует так, как если бы он работал на машине, которая непосредственно воспринимает и выполняет предложения ФОРТРАНА. Набор команд IBM/360 составлял два внутренних слоя:

- команды, допустимые в состоянии “супервизор”;
- команды, допустимые в состоянии “задача”.

В OS/360 состояние “супервизор” используется для работы части управляющей программы, называемой *супервизором*. Поскольку все остальные программы работают в состоянии “задача”, они должны пользоваться только непривилегированными командами. В случае необходимости такие программы при помощи системных макрокоманд обращаются к супервизору. По этим макрокомандам осуществляется вход в супервизор командой обращения к супервизору (SVC).

Все программы OS/360, за исключением супервизора, действовали в состоянии “задача”. Один из фундаментальных принципов проекта заключался в том, что такие программы (компиляторы, сортировки и т.п.) являлись со всех точек зрения проблемными программами и именно так должны трактоваться супервизором. Один и тот же набор средств предлагался и системным, и прикладным программам. В любой момент система состоит из супервизора и всех программ, которые можно получить из подключенных устройств памяти. Так как в установку можно вводить новые компиляторы и прикладные программы, обобщенная машина может расти.

При разработке метода для управления системой второго поколения нужно было согласовать две противоположные точки зрения. В

операционных системах первого поколения считалось, что машина выполняет программы из входящего в нее потока, при этом каждая программа и относящиеся к ней входные данные соответствуют одному применению или проблеме. С другой стороны, в относящихся к первому поколению системах реального времени считалось, что приходящие группы данных отправляются к одной из нескольких программ обработки. Два этих подхода приводили к системам различной структуры. В то же время отличие здесь скорее количественное, чем качественное. Основным здесь является то, что *в обоих случаях программы используются для обработки данных*. Так как именно комбинация программы и данных представляет собой для целей управления элементарную единицу.

OS/360 рассматривала такую комбинацию как отличительное свойство задачи. Рассмотрим, например, программу обработки сообщений и два сообщения А и В. Для того чтобы обработать А и В, в системе создаются две задачи. Одна состоит из программы плюс А, другая из программы плюс В. В этом случае две задачи используют одну и ту же программу, но две разные группы входных данных. Чтобы уяснить ситуацию, приведем еще один пример. Рассмотрим основной файл и две программы X и Y, которые выдают разные сообщения из основного файла. В системе снова создаются две задачи: первая состоит из основного файла плюс X, а вторая – из основного файла плюс Y. В этом случае одни и те же входные данные объединяются с двумя различными программами, и это приводит к образованию двух разных задач.

При определении основных принципов проектировщики OS/360 использовали понятие обработки многих задач. При такой обработке в любой момент времени на системные ресурсы могут претендовать или же использовать эти ресурсы несколько задач. Термин «*мульти-программирование*» применяется обычно в случае, когда один центральный процессор используется одновременно несколькими задачами. Термин «*мультипроцессорная обработка*» используется, когда одна задача решается с помощью нескольких центральных процессоров. Понятие *многозадачной* работы объединяет оба этих случая. Метод оформления работ, которые должны быть выполнены на вычислительной системе, в виде совокупности задач применим без изменений в любом случае.

В OS/360 некоторая совокупность данных, имеющая имя, называется *набором данных*. Набором данных может быть платежный файл, массив статистических данных, программа на языке, транслированная программа, группа предложений для описания работы и т. п. Для хранения наборов данных в системе был предусмотрен архив с каталогом. Архив оказался очень полезен как при создании программ, так и при их исполнении. Программист мог хранить, изменять, перекомпилировать, компоновать и выполнять программы, почти не используя колоды перфокарт.

С точки зрения пользователя, OS/360 состояла из трансляторов, набора сервисных программ и управляющей программы. Более того, с точки зрения управления системой, все прикладные программы, исполняемые на некоторой установке IBM/360, можно было рассматривать как часть ее операционной системы.

В состав OS/360 был введен редактор связей, с помощью которого группа отдельно транслированных программ могла быть объединена в одну рабочую программу. Редактор связей позволял изменять программу, перетранслировав только ту ее часть, в которую внесены исправления. В том случае, когда программа вся целиком не помещалась в доступной области основной памяти, с помощью редактора связей осуществлялась сегментация программы и организация многократного использования областей памяти для размещения сегментов.

Универсальная программа сортировки-слияния могла расположить записи постоянной или переменной длины из набора данных в возрастающей или убывающей последовательности. В процессе обработки в качестве вводной, выводной и промежуточной памяти могли использоваться либо магнитные ленты, либо устройство хранения с прямым доступом. Данная программа учитывала особенности всех тех устройств ввода-вывода, которые ей отводила управляющая программа. Программа сортировки-слияния могла быть использована либо самостоятельно, либо при помощи обращения к ней из других программ, написанных на ряде языков программирования.

В состав сервисных программ входили программы для редактирования, упорядочения и обновления содержимого архива, перестройки индексной структуры архивного каталога, печати каталога, редактирования и пересылки данных с одного носителя на другой.

Управляющая программа OS/360 делилась на главный планировщик, планировщик заданий и супервизор. Выполнение основных функций управления возлагалось на супервизор, который отвечал за распределение памяти, порядок следования задач и управление вводом-выводом. Главный планировщик организовывал общение с оператором. Планировщик заданий выполнял анализ входного потока заданий, распределял устройства ввода-вывода и организовывал для них установочные действия. Кроме того, он начинал и заканчивал выполнение заданий.

Основные действия, выполняемые супервизором были следующие:

- распределение основной памяти;
- загрузка программ в основную память;
- управление параллельным выполнением задач;
- организация службы времени;
- процедуры восстановления после возникновения исключительных условий;
- регистрация ошибок;
- обеспечение итоговых данных об использовании системных средств;

– запуск операций ввода-вывода и управление ими.

Обычно супервизор получает управление центральным процессором в результате *прерывания*. Такое прерывание может произойти либо вследствие явного запроса на обслуживание, либо как машинное прерывание IBM/360, например, прерывание, происходящее при завершении операции ввода-вывода. У супервизора для работы с данными имелось несколько программ доступа. Какие программы доступа можно использовать в некоторый момент, определялось нуждами программы пользователя, структурой конкретного набора данных и типом используемых устройств ввода-вывода.

Задание в OS/360 состояло из одного или нескольких пунктов (шагов) и являлось основной независимой единицей работы. Поскольку каждый пункт задания приводит к выполнению “основной” программы, система формализует каждый пункт как задачу. Затем эта задача с помощью *инициатора-терминатора* (функциональный элемент планировщика работ) ставится в очередь задач. В некоторых случаях выходные данные после выполнения одного пункта передаются другому пункту в качестве входных. Например, с помощью трех последовательных пунктов можно было бы выполнить работу по обновлению файла, сортировке и печати итоговых таблиц.

Планировщик заданий выполнял следующие основные действия:

- чтение заданий на обработку с входного устройства;
- распределение устройств ввода-вывода;
- инициирование процесса выполнения программ для каждого пункта задания;
- запись выходных данных задания.

В своем наиболее универсальном варианте планировщик работ позволял обрабатывать несколько заданий одновременно. На основе приоритетов заданий и доступных в данный момент ресурсов планировщик мог изменять порядок выполнения заданий. Задания могли считываться с нескольких устройств ввода, а результаты – быть записаны на несколько выводных устройств, при этом чтение и запись производились параллельно с внутренней обработкой.

Главный планировщик обеспечивал связь между оператором (сотрудник вычислительного центра) и системой. Оператор с помощью специальных команд мог сообщить системе об изменении состояния устройства ввода-вывода, изменить функционирование системы и запросить информацию о состоянии. Кроме того, оператор использовал главного планировщика для того, чтобы указать планировщику заданий те устройства, с которых вводятся задания, а также для того, чтобы инициировать чтение или обработку заданий.

В общем, управляющая программа выполняла следующие три основные функции:

- управление заданиями;
- управление задачами;

– управление данными.

Две разные, но не независимые проблемы возникли при создании такой системы, как OS/360.

Первая состояла в определении диапазона функциональных возможностей системы. В сущности, это означает определение двух операционных систем:

- одной с максимальными возможностями;
- другой (представляющей собой систему-ядро) с минимальными возможностями.

Вторая проблема состояла в определении набора блоков, из которых можно достаточно эффективно построить как две вышеупомянутые системы, так и системы промежуточного типа. При решении второй проблемы, приводящей к понятию модульности, решающее значение имело требование эффективного использования основной памяти.

Как указывалось ранее, с внешней точки зрения OS/360 представляет собой библиотеку программных модулей. Эти модули являются блоками, из которых может быть создана сама операционная система. В проекте OS/360 при конструировании блоков были использованы три основных принципа, обеспечивающих желаемую степень модульности. Этими хорошо известными принципами являются:

- параметрическая универсальность;
- функциональная избыточность;
- функциональная избирательность.

Параметрическая универсальность заключалась в возможности учитывать изменение от установки к установке количества устройств ввода-вывода, устройств управления и каналов с помощью написания программ с параметрами. Такой способ давно применялся, например, в программах сортировки и слияния, а также в других универсальных программах. В OS/360 этот принцип использовался при генерировании конкретных экземпляров управляющей программы.

Если нужно оптимизировать производительность при наличии двух или более противоречивых целей, то часто наиболее практичным решением является создание двух или более программ, работающих по разным принципам. Этот метод особенно полезен для транслирующих программ, очень чувствительных к такого рода ситуациям. Например, на одной и той же установке может возникнуть необходимость провести одну компиляцию с минимальным использованием основной памяти (даже в ущерб другим целям), а другую – с целью получить максимально эффективную рабочую программу (снова в ущерб другим целям). В том случае, когда противоречивые цели не могут быть согласованы другими способами, создатели OS/360 предусмотрели несколько программ для реализации одной и той же функции – *функциональную избыточность*. Например, имелось два транслятора с языка программирования КОБОЛ.

Для ядра управляющей программы, которое постоянно присутствует в основной памяти, эффективное использование памяти является

особенно существенным. Поэтому каждая функциональная возможность, которая в некоторых установках может не использоваться, оформлялась как необязательная, включаемая по выбору. При генерировании управляющей программы каждая исключенная возможность уменьшала количество основной памяти, требуемое для управляющей программы. Это свойство называлось *функциональной избирательностью*. Наиболее важными из дополнительных возможностей управляющей программы являлись возможности, относящиеся к различным режимам планирования заданий и управления задачами. Имелись следующие возможности такого типа:

- синхронизация задач;
- очереди заданий при вводе и выводе;
- разные методы распределения основной памяти;
- защита основной памяти;
- выбор заданий в соответствии с приоритетами.

Управляющая программа с минимальным набором возможностей осуществляла обычную пакетную обработку заданий. Функционирование центрального процессора и каналов ввода-вывода при этом совмещалось. Программа выполняла следующие функции:

- обнаруживала ошибки и выполняла процедуры восстановления;
- автоматически обрабатывала прерывания;
- выполняла стандартные сервисные программы и работы с данными.

Пункты заданий обрабатывались последовательно по одной задаче в каждый конкретный момент.

Диапазон режимов работы управляющей программы, обеспечиваемый набором дополнительных возможностей, можно указать с помощью двух крайних случаев. В первом, наименее сложном случае, пункт задания мог выполняться параллельно с задачей первоначального ввода (назовем ее А) и задачей вывода результатов (назовем ее В). Поскольку задачи А и В принадлежат управляющей программе, они не соответствуют пунктам задания в обычном смысле. Данная конфигурация в основном предназначалась для того, чтобы уменьшить задержки между обработкой последовательных пунктов – задачи А и В выполняют исключительно функции ввода-вывода.

В другом крайнем случае не более n заданий могли выполняться одновременно, причем параметр n фиксировался в момент генерации управляющей программы. Из различных заданий могли появляться соперничающие задачи. Любая из них могла динамически создать другие задачи, и указать их приоритеты. Было предусмотрено запоминание целого пункта (от задания с самым низким приоритетом) во вспомогательной памяти, если основная память исчерпана. Прерванный пункт возобновлялся, как только ранее занятая область основной памяти становилась снова доступной.

Для того чтобы получить необходимую операционную систему,

пользователь должен был описать имеющуюся конфигурацию ЭВМ, запросить некоторый комплект трансляторов и обслуживающих программ и указать, какие из дополнительных возможностей управляющей программы он хочет получить – все эти действия осуществлялись с помощью специального набора макрокоманд. После того как это было сделано, начинался процесс создания некоторой индивидуальной операционной системы из системной библиотеки OS/360. Такой процесс разбивался на две стадии. Во-первых, ассемблер анализировал макрокоманды и формировал из них поток заданий. На второй стадии с помощью ассемблера, редактора связей и сервисной программы каталога создавалась резидентная часть управляющей программы, заказанный набор трансляторов и сервисных программ.

OS/360 была модульной операционной системой. Она была предназначена для работы в широком диапазоне применений и с разнообразными конфигурациями аппаратуры. Система была “открытой” не только для рассматриваемого класса функций, но и позволяла вводить дополнительные функции.

Конечной целью операционной системы является увеличение производительности всей вычислительной установки; производительность персонала должна рассматриваться наряду с производительностью машин. Хотя в OS/360 нашло отражение много способов увеличения производительности, каждый из них обычно неявно предполагал капиталовложения в некоторую часть установки. Капиталовложения необходимы для добавочного обучения персонала, реализации повышенных требований к памяти или увеличения скорости процессора. Для немногих установок имеется возможность оплачивать любой путь, ведущий к повышению производительности, в большинстве случаев, требования экономии обуславливают тщательно выбранное равновесие между капиталовложениями и отдачей. Модульность OS/360 в значительной степени представляла попытку дать каждой установке возможность найти свою точку экономического равновесия.

При разработке OS/360 одной из основных целей было создание универсального монитора, пригодного и для работы в реальном масштабе времени, и для мультипрограммной работы с внешними устройствами, и для традиционной пакетной обработки. Руководствуясь этой целью, проектировщики решили построить конструкцию более универсальную, по сравнению с известными в то время. После рассмотрения основополагающих понятий проекта мы обсудим элементы OS/360, которые наиболее важны для понимания процесса управления работами и задачами.

В то время считалось, что происхождение основных принципов управления задачами в OS/360 и основное понятие задачи шло от первых систем управления запасами, которые использовали терминалы, работавшие в оперативном режиме. Поэтому для иллюстрации этих принципов целесообразно рассмотреть соответствующие характерис-

тики задачи управления запасами, которую нужно решать в оперативном режиме. Например, в системе резервирования мест на авиалиниях каждый запрос на резервирование места уменьшает запас свободных мест, а отмена запроса увеличивает их. Поскольку ответ агенту авиакомпании должен быть дан в течение нескольких секунд, невозможно накапливать записи для дальнейшей обработки. В другом, противоположном случае, когда файлы обновляются и обрабатываются ежедневно или еженедельно, имеется возможность накапливать и сортировать новые сообщения перед включением их в основной файл.

Рассматривая работу в оперативном режиме, можно сделать три важных вывода:

- каждое сообщение должно обрабатываться как независимая задача;
- так как сгруппировать родственные запросы невозможно, каждая задача тратит относительно много времени на обращения к основному файлу;
- система может получить много новых сообщений до того, как завершится задача, в которой идет обработка старого сообщения.

Отсюда ясно, что нужна управляющая программа, которая могла бы работать в условиях одновременного существования нескольких задач. Всякий раз, когда некоторая задача будет не в состоянии использовать центральный процессор из-за задержек при вводе-выводе, такая управляющая программа должна разрешить использовать его другой задаче.

Другое важное соображение при работе в оперативном режиме связано с размером и сложностью требующихся программ. Действительно, количество команд, необходимых для обработки сообщений, может заметно превосходить объем основной памяти. Кроме того, выбор подпрограмм и последовательность выполняемых действий зависят от содержания входного сообщения. Наконец, подпрограммы, перенесенные в основную память для обработки одного сообщения, могут понадобиться для обработки следующего сообщения. Все эти обстоятельства приводят к тому, что подпрограммы должны вызываться по имени во время счета и быть перемещаемыми (способными размещаться в любом доступном месте основной памяти), кроме того, желательно пользоваться единственной копией подпрограммы для нескольких сообщений.

Основополагающим является тот факт, что задача (работа, которую нужно выполнить над сообщением) должна быть *идентифицируемым и контролируемым* элементом. Чтобы выполнить задачу, требуется некоторая совокупность системных ресурсов:

- центральный процессор;
- подпрограммы;
- место в основной и вспомогательной памяти;
- магистрали для передачи данных во вспомогательную память (т.е. каналы и устройства управления);

– таймер и др.

Поскольку на обладание одним ресурсом могут претендовать несколько задач, существенной функцией управляющей программы является управление системными ресурсами:

- регистрация запросов на ресурсы;
- согласование противоречивых требований;
- приемлемое распределение ресурсов.

На общую схему работы с многими задачами, принятую при проектировании управляющей программы OS/360, сильно повлияли принципы управления задачами, уже проверенные в системах, работавших в оперативном режиме. Однако нет смысла ограничивать понятие “задача” контекстом оперативной обработки сообщений о запасах. Понятие задачи может быть распространено на любую совокупность действий, требуемых от вычислительной системы. Такими действиями могут быть выполнены трансляции, начисление заработной платы или же действия по преобразованию данных.

Чтобы придать законченность этой важной для понимания назначения, состава и структуры современных ОС теме (принципы создания OS/360), приведем еще раз краткие определения терминов, введенных ранее.

С точки зрения учета времени ЭВМ и организации ее действий, основной единицей работы является *задание*. Существенной характеристикой задания является его независимость от других заданий. У одного задания нет возможности повлиять на другое. У программиста также нет средств, с помощью которых он мог бы указать, что одно задание должно зависеть от результата или удовлетворительного завершения другого. Задания описываются с помощью управляющих предложений. Эти предложения могут быть сгруппированы так, чтобы образовать входной поток заданий. Для удобства поток заданий может включать входные данные, однако основной целью его организации является определение и спецификация заданий. Поскольку задания не зависят друг от друга, открыты все возможности для параллельного их выполнения.

Выдавая соответствующие управляющие предложения, пользователь может разбить задание на шаги или *пункты задания*. Таким образом, задание – это совокупность всех работ, связанных с составляющими задание пунктами. В OS/360 пункты отдельного задания были обязательно последовательны – каждый раз мог обрабатываться только один пункт. Кроме того, выполнение пункта могло зависеть от успешного выполнения одного или нескольких предшествующих пунктов. Если заданное условие не выполнено, рассматриваемый пункт может быть пропущен.

Всякий раз, когда управляющая программа распознает пункт (в результате анализа управляющего предложения в задании), она образует для него *задачу*. Задача состоит частично или полностью из

работы, которая должна быть выполнена под управлением программы, названной в пункте задания. Эта программа, в свою очередь, может обратиться к другим программам. У нее имеются для этого две возможности. Во-первых, можно вызвать новую программу в рамках первоначальной задачи. Во-вторых, можно образовать новую задачу. Задача создается (за исключением специального случая начальной программной загрузки) с помощью специальной макрокоманды ATTACH (ОБРАЗОВАТЬ). При инициировании пункта задания макрокоманда ATTACH выдается управляющей программой. В процессе выполнения пункта задания макрокоманды ATTACH могут быть выданы программой пользователя.

С точки зрения управляющей программы, все задачи являются независимыми в том смысле, что они могут выполняться одновременно. Однако между задачами, которые относятся к одному заданию (они относятся к одному и тому же пункту задания), могут существовать внутренние зависимости, определяемые логикой программ. Для того чтобы учесть эти зависимости, в системе предусмотрены средства, с помощью которых задачи одного и того же задания могут быть синхронизированы и иерархически упорядочены. В результате одна задача может ждать в указанной точке окончания выполнения другой задачи. Аналогично задача может ждать завершения подзадачи (задачи, расположенной в иерархии на уровень ниже). Кроме того, задача может ликвидировать подзадачу.

Хотя в потоке заданий может находиться много заданий, каждое из которых состоит из многих пунктов, порождающих, в свою очередь, много задач, тем не менее, можно себе представить несколько совершенно реальных вырожденных случаев. Например, в случае управления запасами в оперативном режиме все вычислительные средства могут быть отданы одному заданию, состоящему из одного-единственного пункта. В каждый момент времени этому пункту задания может соответствовать много задач, по одной на каждое сообщение. С другой стороны, совершенно естественно ожидать, что на многих установках все задания имеют по несколько пунктов (например, компиляция – редактирование связей – выполнение), причем среди пунктов нет таких, которые порождают несколько задач.

В большинстве случаев программы, которые предстоит выполнить, и подлежащие обработке данные не представляют собой новой для системы информации. Они уже имеются в системе как результат выполнения предшествующих заданий. Такие программы и данные не нужно задавать для задания снова. Вполне достаточно, чтобы эти данные были названы (идентифицированы) в управляющих предложениях, вставляемых в поток заданий на место таких данных. Поток заданий состоит из подобных управляющих предложений и иногда из данных, являющихся новыми для системы. Управляющее предложение – это первое предложение каждого задания OS/360 (JOB). В таком

предложении могло быть указано название задания, учетный номер и фамилия программиста. Кроме того, могло быть указано, к какому из классов относится задание по своему приоритету, могут быть определены различные условия, в случае невыполнения которых система пропускает остающиеся пункты.

В заключение необходимо еще раз подчеркнуть, что концепция, положенная в основу OS/360, сыграла ключевую роль в современной концепции операционных систем. Именно по этой причине ей уделено так много места в данной юните.

2.1.3. ДОС – операционная система для персональных ЭВМ

Появление и развитие персональных ЭВМ вызвало необходимость создания для них операционных систем, которые (вначале) должны были соответствовать их ограниченным ресурсам. Далее будут рассматриваться принципы работы операционной системы ДОС только для ПЭВМ на базе процессора Intel 80x86. Такие ПЭВМ наиболее распространены в России.

Одной из первых ОС для ПЭВМ была CP/M – система, рассчитанная на применение в микропроцессорных ЭВМ, используемых в режиме одиночного пользователя. В этой системе уже имелись средства ведения файлов, проверки и обновления содержимого памяти, а также средства доступа к ассемблеру и компиляторам.

Первые ПК создавались на знаниях и интересе любителей. ПК развивались вместе с развитием микрокомпьютерной технологии от домашних игрушек до серьезных систем. Программное обеспечение и операционные системы развивались вместе с компьютерами. Ранние операционные системы представляли собой немногим более, чем командные процессоры Бейсика или Паскаля, которые обеспечивали поддержку языков и простое управление файлами. Эти системы обычно ориентировались на конкретный компьютер и были несовместимы с другими. Когда компьютеры начали ориентироваться на массовое применение, операционные системы начали стабилизироваться. Среди них были Apple-DOS, NorthStar DOS и CP/M. Все они имели похожие характеристики:

- однопрограммный однопользовательский режим работы;
- поддержка файлового каталога;
- поддержка интерпретаторов и компиляторов языков программирования;
- несовместимость программ и данных между системами.

CP/M была создана как открытая ОС, предназначавшаяся для поддержки создания программ для Intel, но эта ОС не ориентирована на какую-либо модель или марку компьютеров. CP/M состояла из:

- базовой системы ввода-вывода (BIOS);
- базовой дисковой операционной системы (BDOS);

– командного процессора (CCP).

BIOS ориентирована на работу с аппаратурой компьютера и предназначена для управления дисплеем, принтером и дисковой системой. Путем написания соответствующей BIOS производители компьютеров могли использовать приемлемую и развитую операционную систему с множеством доступных программ. Благодаря своей адаптируемости CP/M стала промышленным стандартом ОС для ПК на микропроцессорах Intel 8080 или Z80.

Микропроцессоры 8080 и Z80 могли адресовать только 64К памяти, поэтому CP/M строилась как однопользовательская однопрограммная ОС. Была построена и многопользовательская версия, названная MP/M, но она не пользовалась популярностью, в основном из-за медленной работы (8080 не очень быстрый процессор), и потому, что ПК не особенно подходят для многопользовательской работы.

Наибольшее количество пользователей в мире безусловно использовало и еще использует систему ДОС – сокращение от Дисковой Операционной Системы, поэтому мы уделяем ей особое внимание.

ДОС, управляющая работой ПЭВМ, является адаптацией ОС, созданной для использования на 8086, и заметно похожей на CP/M. Microsoft купила ее у создателя, а IBM купила лицензию у Microsoft. ДОС состоит из трех модулей, схожих по функциям с BDOS, CCP и BIOS в CP/M, а пользовательский интерфейс почти идентичен используемому в CP/M.

У ДОС появляются дополнительные возможности, такие, как каналы, фильтры, переназначение ввода-вывода, пометка файлов временем и датой и иерархическая структура файлового каталога.

Первые компьютеры IBM PC были очень похожи на своих предшественников с микропроцессорами 8080 и Z80 – у них было 64К памяти, флоппи-диски и процессор. Однозадачная ДОС была подходящей и адекватной этим компьютерам, но PC имели три архитектурные особенности, предназначенные для расширения. Микропроцессор 8088 имел:

- адресуемую до 1М память;
- векторную систему прерываний;
- клавиатура и дисплей IBM PC являлись составной частью компьютера, в отличие от видеотерминалов, присоединенных через последовательный порт.

В минимальном варианте ДОС состоит из трех файлов: MSDOS.SYS, IO.SYS и COMMAND.COM. Первые два файла имеют атрибуты “скрытый”, “системный” и не высвечиваются на экране в обычном режиме.

Файл MSDOS содержит программы, реализующие функции операционной системы. В файле IO.SYS содержатся программы обмена информацией с внешними устройствами. Последний из файлов – COMMAND.COM – интерпретатор команд. Именно он принимает информацию с клавиатуры и расшифровывает ее смысл. Данные три

файла позволяют работать с ограниченным подмножеством команд ДОС – внутренними командами.

Все команды ДОС делятся на две группы – внутренние и внешние. Внешние команды представляют собой отдельные файлы, которые можно стирать, копировать и т. п. Таким образом, ДОС для пользователя – это *ядро из трех файлов плюс оболочка из файлов внешних команд*.

В этой оболочке есть еще один слой – драйверы устройств. Драйвером называется специальная программа, предназначенная для обслуживания внешнего устройства. Само существование драйверов связано с наличием различных типов аппаратных компонентов, например экранов различного типа и с различным разрешением.

Дальнейшее развитие оболочечного принципа построения операционных систем породило понятие операционной оболочки, т.е. системной среды открытого типа, состав которой можно легко дополнить необходимыми пользователю прикладными и сервисными программами.

Оперативное Запоминающее Устройство (ОЗУ) – внутренняя память, в которой содержатся операционная система, коды исполняемой программы и данные. В ранних моделях персональных компьютеров размер адресуемой памяти, т. е. *наибольший размер ОЗУ, с которым может работать процессор*, составлял 640К. Общая структура адресуемого пространства была следующей:

256К – ПЗУ служебная память;

128К – экранная память;

640К – ОЗУ, память для ДОС и пользователя;

итого общий размер адресуемого пространства памяти – 1М.

Рассмотрим, что происходит в операционной среде после включения питания.

Вначале активируется Базовая Система Ввода-Вывода (BIOS), которая хранится (прошита) в определенном участке постоянной памяти (ПЗУ). BIOS выполняет комплекс программ начального тестирования компьютера. На экране этот процесс, как правило, отражается в виде возрастающей последовательности цифр, означающих количество килобайт проверенной памяти. Отработав, тестирующая система должна передать управление собственно операционной системе.

Для этого надо загрузить файлы операционной системы. В ДОС было предусмотрено соглашение – файлы операционной системы всегда берутся с диска А, а если в дисководе А нет дискеты, то файлы будут читаться с диска С. Необходимость такого соглашения вызвана тем, что в случае какой-либо неисправности на жестком диске можно запустить операционную систему с дискеты и проверить причину неисправности. Впрочем, имеется возможность настройки конфигурации последовательности обращения к внешним устройствам при загрузке системы.

ДОС, получив управление, ищет файл CONFIG.SYS, который содержит информацию о драйверах устройств и, загрузив драйверы получает

систему нужной конфигурации. Затем ДОС загружает командный процессор COMMAND.COM.

Командный процессор ищет файл AUTOEXEC.BAT и выполняет команды этого файла. После чего командный процессор выдает подсказку ДОС и ждет команду пользователя.

2.1.4. Система прерываний персональных ЭВМ

В персональных ЭВМ имеются более двух сотен прерываний, которые образуют **систему прерываний** – комплекс технических и программных средств, обеспечивающих возможность прерывать выполнение программы при поступлении внешних или внутренних сигналов прерывания и продолжить выполнение прерванной программы от точки прерывания после обработки сигналов прерывания.

Прикладные программы, предназначенные для решения задачи или класса задач в определенной области применения систем обработки данных, используют прерывания, чтобы получить доступ к системным ресурсам. Получив сигнал прерывания, процессор выполняет операцию, состоящую в регистрации состояния процессора, предшествовавшего прерыванию, и установлении нового состояния. **Прерывание** – это реакция процессора на некоторое условие, которое возникло в процессе или вовне его и позволяет обработать возникшее условие специальной программой, чтобы затем вернуться к прерванной программе.

Имеются следующие основные виды прерываний:

прерывание ввода-вывода – внешнее прерывание, инициируемое устройствами ввода-вывода при изменении их состояния, например, завершение операции ввода-вывода, ошибка ввода-вывода, готовность аппаратуры;

прерывание по защите памяти – программное прерывание, обусловленное попыткой обратиться к данным, размещенным в области памяти, недоступной данной программе;

прерывание по сбою – один из типов прерываний, возникающий при появлении сигнала от схем контроля ЭВМ, если разряд маски контроля машины равен единице;
и др.

Каждый тип прерывания обладает определенным **приоритетом** – характеристикой, определяющей его права на предоставление ресурсов. **Приоритет прерывания** – это числовое значение (ранг), приписываемое прерыванию определенного типа. При одновременном поступлении нескольких прерываний обслуживается прерывание с большим приоритетом. В принципе, прерывания – это готовые процедуры, которые вызываются для выполнения определенной задачи управления процессами в системе.

В ПЭВМ (типа IBM PC) различные прерывания имеют номера от 0 до 0xff (шестнадцатичное число). Некоторые из них определены для

использования процессором. Например, прерывание 0 возникает при делении на 0. Другие определены для вызова функций BIOS, третьи – для использования ДОС. Для каждого из этих трех архитектурных слоев определен свой набор прерываний. Оставшиеся прерывания доступны для использования прикладными программами и программами обслуживания устройств.

Каждое прерывание представлено в памяти четырехбайтным значением адреса, которое называют *вектором*. Эти значения располагаются в памяти со смещениями от 0 до 0x3ff. При прерывании содержимое регистра признаков и четырехбайтный адрес выполняемой команды сохраняются в стеке. После этого прерывания запрещаются, и начинает выполняться программа с адреса, соответствующего происшедшему прерыванию. Эта программа должна сохранить используемые ей регистры, выполнить свою задачу, восстановить значения регистров, и выполнить команду возврата из прерывания, которая восстанавливает адрес прерванной программы и регистр признаков, так что прерванная программа продолжит свое выполнение с того места, где была прервана.

Аппаратные прерывания вызываются событиями, физически связанными в аппаратуре с соответствующими векторами прерываний. Например, клавиатура связана с прерыванием 9. Нажатие клавиши вызывает прерывание выполняемой программы, как было описано выше, и переход по адресу, находящемуся в векторе прерывания, соответствующему прерыванию 9. В памяти этот вектор находится по адресу 0x24.

Программные прерывания происходят при выполнении в текущей программе команды INT с номером прерывания в качестве операнда. В остальном нет никакой разницы между программным и аппаратным прерыванием.

Для управления выполнением программ в ДОС используется Program Segment Prefix (PSP). PSP – это управляющая область в 256 байт, которая строится в памяти в начале каждой программы. Она содержит различные поля, используемые ДОС для управления выполнением программы. Приведем содержание некоторых из этих полей:

1) Сегментный адрес верхней границы памяти. При выполнении программы ДОС выделяет ей участок памяти, в который программа загружается. Это поле содержит сегментный адрес конца этого участка памяти.

2) Адрес обработчика завершения. При выполнении программы ДОС запоминает текущее содержание вектора прерывания 0x22 в этом поле. После завершения программы ДОС восстанавливает значение, используя это поле. Вектор прерывания 0x22 указывает на системный обработчик завершения программ.

3) Сегментный адрес PSP родителя. Любая программа выполняется в результате обращения другой программы к ДОС. Обычно программой-

отцом является командный процессор ДОС (COMMAND.COM), хотя любая программа может быть родителем любой другой. Это поле в PSP содержит сегментный адрес PSP программы-отца. У командного процессора нет “отца”, поэтому это поле в PSP командного процессора содержит сегментный адрес собственного PSP, что является указателем на самого себя.

4) Таблица указателей файлов. Это поле представляет собой массив, каждый элемент которого представляет указатель файла. При открытии файла в программе ДОС возвращает в программу номер файла для использования при обращении к ДОС для записи в файл и чтения из него. Эти номера файлов – номера байт в таблице указателей файлов, а элементы этой таблицы хранят значения, указывающие на соответствующую файлу структуру в системных таблицах управления файлами.

5) Сегментный адрес области системных параметров. Это поле содержит сегментный адрес области системных параметров, создаваемой ДОС для выполняемой программы. Область системных параметров – это выделенный задаче участок памяти, который может быть освобожден, если значения параметров не используются.

6) Адрес стека на время вызова функции ДОС. В это поле ДОС записывает значения регистров сегмента стека и указателя стека текущей программы при вызове из программы функции ДОС. Затем ДОС переключается на свой собственный стек. Перед возвратом в вызвавшую программу ДОС использует эти значения для восстановления значения этих регистров.

У каждой программы есть свой PSP. Но ДОС известен только один PSP – находящийся перед программой, запущенной последней. Программа в ДОС может порождать выполнение другой программы, и программы-дети могут наследовать значения из PSP “отца”, но ДОС знает только об одной активной задаче и только об одном PSP.

Программа может открыть несколько файлов одновременно. С каждым файлом ассоциируется указатель, являющийся элементом массива в PSP. К файлам программа обращается по номерам указателя в массиве, а в элементе массива с таким номером хранится ссылка на соответствующую структуру в системных таблицах управления файлами.

PSP активно используется при организации режима многозадачности.

2.1.5. Режим многозадачности

Первоначально ДОС была сделана для поддержки работы только одной задачи. ДОС организует загрузку и выполнение задач и выполняет запросы на ввод-вывод. Она управляет дисковыми каталогами и файлами, работает с системными часами, выводит данные на печать, консоль и возвращает программе символы, введенные с клавиатуры. ДОС – это в сущности сервер, обслуживающий иерархическую файловую

систему и устройства, ориентированные на запись. ДОС обеспечивает одному пользователю выполнение одной задачи. И с этой службой ДОС справляется.

После первоначальной загрузки ДОС память размером в 640К, имеющаяся в ПЭВМ, распределена следующим образом:

- память с адресами от 0 до 0x400 зарезервирована для векторов прерываний;
- за ними следует программа ДОС;
- затем идут драйверы устройств, загруженные вместе с ДОС, например, при использовании виртуального диска или драйвера терминала ANSI, программы этих драйверов располагаются после ДОС;
- после драйверов идет резидентная часть командного процессора;
- нерезидентная часть находится после резидентной части командного процессора.

Запущенная пользователем из командной строки, программа загружается в нерезидентную часть, в конце которой находится область нерезидентной части командного процессора. Пользовательская программа может использовать эту область. В этом случае резидентная часть командного процессора подгружает нерезидентную после окончания программы.

После загрузки командным процессором программы она начинает выполняться. При необходимости обращения к ДОС, например для выполнения операции с файлом, вызов ДОС осуществляется программным прерыванием с передачей параметров через регистры. В зависимости от параметров ДОС выполняет одну из своих функций. Вызвавшая ДОС программа находится в состоянии ожидания. Результаты функций ДОС возвращаются в регистрах.

Такая последовательность событий описывает работу типичной однозадачной ОС. Ранние версии ДОС не обеспечивали одновременное нахождение в памяти и выполнение пользовательских программ или поддержание информации о более чем одной задаче в памяти. Единственный метод сделать несколько задач активными в памяти – оформить несколько задач как программы обработки прерываний. Для ДОС эти программы будут выглядеть как ISR-программы (программы обработки прерываний), поддерживающие выполнение единственной задачи. В ДОС задаче разрешалось порождать подчиненную подзадачу, но только одна из них может быть активной в текущее время. Главная задача будет бездействовать, пока не завершится подзадача.

В первых версиях ДОС было одно исключение. Создатели ДОС включили в состав ДОС программу печати со спулингом по имени PRINT.COM. Спулинг – это слово, означающее “одновременное оперативное выполнение периферийных операций”. Спулинг при печати позволяет печатать файл, в то время как пользователь выполняет на

компьютере другую задачу, для которой не требуется принтер. PRINT.COM является резидентной программой, остающейся в памяти, поддерживающей очередь запросов на печать, и печатающей заданные файлы. Одновременно с этим возможности компьютера и ДОС остаются доступными пользователю. Наличие программы, читающей дисковые файлы, читающей и записывающей в файл очереди на печать и переводящей страницы на принтере, пока пользователь делает что-нибудь еще, позволял предположить, что в ДОС реализуется некоторая мультизадачность без объяснения, как это делается.

Это предположение и пытливая натура поколения хэккеров, конечно, подталкивали к открытию возможностей ДОС по ограниченной мультизадачности. Некоторое время ключи не удавалось подобрать; затем те, кто понял технику написания функционально полных резидентных (TSR) программ, хранили секрет, так как они продавали эти программы. Но остальные, однако, начали “взламывать” чужие программы и открыли секрет.

Режимы многозадачности в полной мере были реализованы в следующих поколениях операционных систем – UNIX, Windows и др.

В основе режима выполнения нескольких задач одновременно может лежать один из двух следующих способов:

- использование механизма периодических прерываний, которые и будут осуществлять переключение выполняемых задач;
- использование программы, которая сама будет передавать контроль процессам через достаточно регулярные интервалы времени.

О первом способе, обычно называемом многозадачным режимом с *вытесняющими задачами*, говорят иногда, как об истинном мультипрограммировании, поскольку такая система будет работать даже, если процессы выполняют очень длинные операции или происходят ошибки типа бесконечного закликивания. Второй способ, который называют *совместным мультипрограммированием*, является более эффективным по двум причинам:

- задача переключения процессов становится немного проще;
- маловероятно, чтобы процессы тратили свою часть ресурсов процессора на холостые циклы, ожидая ввод.

В любом случае, чем меньше времени ОС расходует на переключение процессов, тем больше времени она может дать прикладной программе. Чтобы выполнить переключение двух процессов, система с вытеснением должна сохранить все регистры процессора для вытесняемой задачи и загрузить все хранимые регистры процессора вытесняющей программы. Совместная система может исключить процесс с сохранением и перезаписью некоторых регистров (программных, стековых указателей), поскольку операция переключения контекста скрывается в этом случае за операцией ввода-вывода, и задачи, как правило, не ждут значений регистров.

Подход к реализации *совместного мультипрограммирования*, при котором переключение контекста сводится к простому переприсваиванию указателей и вызове функций, основан на идее конечных автоматов (finite-state machines – FSM). Благодаря маленьким размерам и эффективности, FSM широко используются для взаимодействия, в контроллерах, для самой разнообразной текстовой обработки. Они точно кодируют состояние системы и переходы состояний таким образом, что значительно облегчает контроль системы.

Основная идея FSM состоит в том, что система находится всегда в одном из нескольких фиксированных состояний. Каждое состояние обрабатывается частью программы, отвечающей за организацию вывода (в зависимости от состояния) и (или) за изменение состояния процесса при вводе.

Состояние необходимо изменять тогда, когда обработчик текущего состояния завершает свою работу и возвращает контроль FSM, который, в свою очередь, запустит обработчика нового состояния. Например, “Основное состояние” эмулятора терминала ANSI (стандарт представления символов) просто отображает на экране все вводимые символы, пока не будет нажата клавиша Esc. После этого система попадает в новое состояние “Входная команда”, которое ожидает символ “[“. Если это состояние действительно получит требуемый символ, вводится новое состояние “ANSI команда”, при котором ожидается либо численный аргумент, либо команда ANSI. Если состояние “Входная команда” получает при вводе символ “[“, то оно выдает на экран символ Esc (для состояния “Основное состояние”) и введенную литеру; затем система возвращается в “Основное состояние”.

Внешне этот подход похож на обычное структурное программирование. Основная программа сводится в данном случае к простому механизму, который по мере необходимости запускает обработчика соответствующего состояния, а все необходимые действия выполняет сам обработчик.

Возможен и альтернативный вариант – создать набор переменных несвязанных состояний. Тогда для каждого вводимого символа надо было бы проходить через ряд вложенных условных if-выражений, чтобы определить, является ли этот символ простым вводом, который необходимо отобразить на экран, либо это часть команды, либо конец командной строки, которая должна быть выполнена.

Имея такое дерево решений, трудно гарантировать надлежащую обработку каждого варианта, равно как и добавление новых.

При работе с FSM точное задание каждого состояния намного облегчает проверку того, делается ли именно то, что надо делать именно в этом состоянии. Кроме того, в FSM проще включить новое состояние, чем в дерево решений. Набор всех состояний чаще всего представляется в виде перечня состояний, и небольшая FSM реализуется обычно как простой переключатель на поле текущего состояния. Поскольку

длинные тексты переключателей трудны для работы, в больших FSM программы обработчиков состояний выделяют в отдельные функции.

Отсюда содержимое регистров ЭВМ, памяти и внешних устройств – это состояние и текущее состояние всегда определяют действия и будущие состояния. FSM всегда знает, что делать, и она не должна это вычислять. Вытекающая отсюда эффективность является главной причиной того, что FSM используют, как правило, для обработки прерываний, при которых обработчик должен продолжительное время хранить контекст процесса и в то же время возвращать контроль прерванному коду как можно быстрее.

Если собрать глобальные переменные всех задач в записи задач, можно организовать режим многозадачной машины (ММ), просто изменяя указатели задач во время смены состояний.

Изменение состояния и вызов функций не только более быстрые операции, чем сохранение и восстановление регистров и загрузка стеков, но они также очень экономно используют память. Традиционные подходы реализации мультипрограммирования требуют также, чтобы каждая задача имела свой собственный стек для вызовов процедур и для хранения локальных переменных.

Например, DOS (начиная с 4-й версии), допуская несколько задач, каждой из которых требуется стек объемом 16 К, выделяет стекам половину всей доступной памяти, и большая часть этой памяти почти не используется. ММ может работать с единственным стеком процессора, поскольку задачи никогда не попадают в состояние бесконечного ожидания. При вызове функции обработки текущего состояния задачи всегда возвращают контроль ядру системы.

Эффективность работы при использовании ММ достигается за счет того, что прикладная программа разбивается на набор обработчиков состояний.

Большинство прикладных программ, как правило, работает в диалоговом режиме. Большую часть времени они тратят на циклы, в которых ожидают получить информацию пользователя определенного типа, обрабатывают ее и снова ждут данные пользователя. Для работы с ММ требуется, чтобы при написании прикладной программы одно состояние поддерживало бы режим диалога (высветить подсказку, панель диалога или что-либо подобное), а второе состояние обрабатывало бы ответ пользователя.

Самые длинные неинтерактивные операции являются итерационными. Такие операции вполне естественно можно превратить в серию состояний. Например, процедура поиска в БД является последовательностью состояний Получить запись/Проверить запись.

Разбиение программы на конкретные именованные состояния намного облегчает реализацию системы контекстной помощи. ММ всегда будет поддерживать контекст автоматически, если перед каждой подсказкой задать переменные контекста.

Традиционные схемы мультипрограммирования требуют от программиста обращать внимание на следующие проблемы. Если несколько задач делят между собой файлы, программист должен использовать некоторую систему блокировки (замыкания) для того, чтобы задача А не могла обратиться к данным, исправленным задачей В и еще не сохраненным в общей структуре данных. Эта проблема стоит не на последнем месте по важности в системах с вытесняемыми программами, которые могут передавать контроль даже в середине оператора на высокоуровневом языке при каждом вызове системных процедур.

В ММ состояния являются автомными. Никакая задача не может получить время процессора, пока не закончит работу обработчик состояния текущей задачи. Следовательно, общие структуры данных можно многократно (в пределах одного состояния) изменять и не опасаться при этом, что другие задачи используют неправильные данные. Таким образом, исчезает потребность в системах блокировки данных и значительно уменьшается вероятность борьбы за ресурсы, нередко приводящая к “клинчу” (когда две задачи захватывают по одному ресурсу из двух нужных им обоим и повисают, поскольку не могут получить второй ресурс).

Поскольку многозадачные системы характеризуются высокой скоростью и низкими затратами памяти, они желательны везде, где прикладным программам требуется и мультипрограммирование, и высокая эффективность.

В целом, описанные мультипрограммные машины являются концептуально простым подходом к реализации многозадачного режима, который прост в использовании и достаточно эффективен.

2.1.6. Драйверы операционной системы

Важнейшая задача любой операционной системы состоит в обеспечении средств взаимодействия с различными типами устройств вычислительного комплекса для прикладных программ и для нужд самой операционной системы.

Для того чтобы прикладная программа могла взаимодействовать с внешним устройством, операционная система должна удовлетворять двум основным требованиям.

Во-первых, должен существовать определенный интерфейс между прикладной программой и операционной системой. Этот интерфейс должен быть достаточно гибким, чтобы прикладная программа могла точно определить свои действия при работе с требуемым устройством.

Во-вторых, операционная система обязана уметь передавать и принимать данные от устройства и управлять его работой.

Такой интерфейс в ДОС обеспечивается так называемыми драйверами устройств. В первых версиях ДОС обеспечивать поддержку

дополнительных устройств после покупки было довольно затруднительно. Операционная система не содержала функциональных запросов прикладного уровня для нестандартных устройств, и сами драйверы были глубоко запрятаны в BIOS. Добавление или изменение драйвера устройства требовало корректировки исходных кодов BIOS (при их наличии, разумеется), повторного ассемблирования и копирования полученных кодов на загрузочную дорожку системного диска. Очень часто для выполнения указанных операций не было даже соответствующих утилит. Более того, такие компьютеры, как IBM PC не позволяли и этого, т.к. их BIOS записана в ПЗУ (постоянном запоминающем устройстве). Изменение содержимого ПЗУ требует наличия специального программатора – устройства, которое записывает информацию в программируемое ПЗУ.

Вероятно, самым значительным нововведением в операционных системах ПЭВМ, с тех пор как появилась CP/M, стало то, что MS-DOS версии 2.0 и выше стали обеспечивать не только возможность установки драйверов, но и стандартный расширяемый интерфейс, который дает программам возможность взаимодействовать с драйверами. В результате значительно возросло количество устройств, поддерживаемых ДОС, и появились драйверы псевдоустройств, обеспечивающие системы такими средствами, как высокоуровневые графические интерфейсы.

Драйвер устройства в ДОС – это подпрограмма, которая вызывается ДОС, с одной стороны, и взаимодействует с конкретным устройством, с другой. Как посредник между системой и аппаратурой, драйвер устройства передает данные между программой и устройством.

Драйверы устройств решают две основные задачи. Первая заключается в обеспечении стандартного интерфейса со всеми программами, желающими использовать определенное устройство, независимого от конкретных особенностей устройства. Программа, выполняющая обработку текста, или электронная таблица, производящая вычисления, может не заботиться о типе терминала, подключенного к системе, выдавая простые команды типа “Отобразить символ” и “Получить символ”. Все технические детали по пересылке символов берет на себя драйвер, обеспечивая тем самым необходимый для прикладной программы высокоуровневый интерфейс. Замена терминала может вызвать замену драйвера, но при этом в прикладной программе не потребуются делать никаких изменений.

Драйверы дисководов должны обеспечивать стандартный интерфейс для всех используемых типов дисков, при этом программа, осуществляющая ввод/вывод с диска, будет работать с дискетой любого формата, с жестким диском и даже с псевдодиском в ОЗУ, не замечая никаких различий. То есть первая задача драйвера состоит в обеспечении независимого от устройства унифицированного интерфейса.

Второе целевое назначение драйверов устройств заключается в том, что они для всех прикладных программ обеспечивают сервис, подобный библиотекам функций времени выполнения. Любая программа освобождена не только от необходимости поддержки множества разноформатных устройств, но и от необходимости поддерживать вообще какие-либо форматы. Все заботы по поддержке устройств возложены на драйверы устройств. В связи с тем, что все драйверы собраны в операционной системе, требуется лишь одна копия каждого драйвера.

В результате этого программы, написанные с использованием интерфейса, предоставляемого ДОС, вообще не содержат в себе драйверов. В операционной системе драйверы могут быть добавлены для того, чтобы заменить встроенные драйверы системы, т.е. пользователь может написать свой собственный драйвер. Как подчеркивалось выше, прикладные программы при этом ничего не “замечают”.

Могут быть созданы драйверы, не поддерживающие реальных устройств. Например, можно написать драйвер, который поддерживает несуществующее устройство – эмулятор диска в ОЗУ. Такие устройства получили название “виртуальные устройства”, а драйверы таких устройств, соответственно, “драйверы виртуальных устройств” или просто “виртуальные драйверы”.

Реальные или виртуальные устройства не ограничены, по сути дела, только операциями ввода-вывода. На драйвер может быть возложена любая функция по преобразованию данных. Высокоскоростные процессоры для выполнения больших объемов вычислений с плавающей точкой – это только один из примеров устройства преобразования информации. Кроме того, драйверы могут программно эмулировать реальные устройства, которые отсутствуют в конкретной системе, такие, как часы или сопроцессор с плавающей точкой.

Основное правило, при каких условиях некоторую функцию следует удалить из программы и перенести в драйвер, состоит в том, что если какая-либо функция выполняет ввод-вывод на физическом уровне (т.е., работая непосредственно с аппаратурой), то эта функция – кандидат для переноса в драйвер. Выделение программ-обработчиков операций ввода-вывода в драйвер устройства порождает четыре следствия:

- делает программы переместимыми;
- делает обработчики операций ввода-вывода доступными для других программ, желающих работать с этим устройством;
- увеличивает в размерах систему;
- замедляет время доступа к аппаратуре.

Некоторое увеличение размера памяти, занимаемой системой, не имеет большого значения, а вот увеличение времени доступа может быть критическим фактором для некоторых приложений. Когда принимается решение о написании драйвера, необходимо тщательно

взвесить скоростные характеристики программы, с одной стороны, и повышение совместимости программ и доступность драйвера, с другой стороны. Увеличение времени доступа за счет накладных расходов при каждом обращении к драйверу более заметно для устройств, которые передают за один раз слово или байт данных. В драйверах, передающих за одно обращение целый блок данных, накладные расходы заметно уменьшаются.

Установку и замену драйверов осуществляют в процессе начальной загрузки системы. Процесс начальной загрузки ДОС начинается со сброса системы. Аппаратура системы устанавливается в состояние сброса при включении питания компьютера. Сразу после сброса процессор начинает выполнять команды, находящиеся в самом конце его адресного пространства. По этим адресам находится ПЗУ, содержащее начальный загрузчик, задача которого заключается в загрузке системной области диска в память

Системная область диска, загружаемая в память начальным загрузчиком, называется вторичным загрузчиком. В случае ДОС, работающей на IBM-совместимом компьютере, это самый первый сектор диска длиной 512 байт. Такой маленький размер объясняется тем фактом, что BIOS находится в ПЗУ. Вторичному загрузчику в этом случае для загрузки остальной части системы достаточно обратиться к BIOS, которая всегда находится в ПЗУ. В системах, не содержащих BIOS в ПЗУ, начальный загрузчик должен считывать с диска программу, способную обеспечить возможность вторичному загрузчику считать остальную часть системы. В таких системах начальный загрузчик должен считывать довольно большую часть диска.

Сама ДОС загружается только после того, как будет считан в память вторичный загрузчик. Именно по этой причине возможен запуск игр, не требующих для своей работы ДОС, или загрузка других операционных систем. Собственно, тип загружаемой системы зависит от того, что именно считывается с загрузочного диска. При загрузке ДОС вторичный загрузчик предполагает наличие на диске корневого директория и, как минимум, двух системных файлов (MSDOS.SYS и IO.SYS см. выше). После того, как вторичный загрузчик находит и загружает эти файлы, начинается процесс инициализации ДОС. Интерфейсный файл IO.SYS содержит стандартные драйверы, которые будут использоваться при инициализации и работе ДОС.

Сама процедура инициализации заключается в распределении частей ДОС в памяти, создании всех внутренних таблиц, рабочих областей и т.п., и, наконец, инициализации всех устройств, связанных с системой. Инициализация устройств заключается в посылке команды INIT каждому из драйверов, содержащихся в интерфейсном файле. После инициализации устройств процедура инициализации заканчивает создание внутренних таблиц, и система к этому моменту готова к работе. До окончательного завершения, однако, остается еще один шаг.

В этой точке процедура инициализации проверяет наличие файла CONFIG.SYS. Если указанный файл отсутствует, то ДОС загружает стандартный интерпретатор команд и передает ему управление. Если же файл CONFIG.SYS найден, то выполняется еще один шаг инициализации. На этом этапе предоставляется возможность подключить к ДОС собственные пользовательские драйверы устройств.

Файл CONFIG.SYS – это обычный текстовый файл, который должен быть расположен в корневом директории диска, с которого происходит загрузка системы (если этот файл находится не в корневом директории, то процедура инициализации предполагает, что он совсем отсутствует). Файл CONFIG.SYS содержит команды, руководствуясь которыми процедура инициализации изменяет и/или дополняет стандартную конфигурацию ДОС. Если этот файл доступен, процедура инициализации (но не COMMAND.COM – он еще не загружен) считывает его в память и обрабатывает строка за строкой. Каждая строка содержит одну команду конфигурации. В данном случае наиболее важна команда DEVICE, которая имеет следующий формат:

DEVICE=[d:][path]filename[.ext][parameters],

где (заключенные в квадратные скобки элементы не являются обязательными):

d: – идентификатор дискового,
path – путь к драйверу,
filename – имя файла, содержащего драйвер,
ext – расширение имени файла,
parameters – параметры для драйвера.

Эта команда задает необходимость установки нового драйвера. Программа драйвера, содержащаяся в заданном драйвере, похожа на обычную COM программу, но имеет некоторые специфические особенности. В общем случае, драйвер представляет собой особую форму резидентной программы. Когда в файле CONFIG.SYS встречается команда DEVICE, соответствующий драйвер загружается в память и анализируется. Заголовок драйвера содержит информацию о типе, имени, атрибутах устройства и определяет точки входа в программу. После загрузки драйвера ДОС обращается к драйверу с командой INIT.

Драйвер выполняет инициализацию и возвращает управление ДОС, указывая адрес конца драйвера, т.е. адрес первого свободного байта памяти, непосредственно следующего за драйвером. На этом установка драйвера заканчивается. Указание адреса конца драйвера при возвращении управления ДОС после выполнения команды INIT подобно указанию размера памяти, занимаемой программой, при вызове функции ДОС “Остаться резидентом”. По возвращаемому адресу ДОС определяет расположение свободной памяти. Если файл CONFIG.SYS содержит другие команды DEVICE, следующий драйвер загружается непосредственно после предыдущего.

После того, как обработка файла CONFIG.SYS закончена, загружается еще один драйвер – драйвер фиктивного устройства (NUL-драйвер). Затем ДОС завершает инициализацию загрузкой постоянной части COMMAND.COM или другой, определяемой пользователем оболочки.

При загрузке драйверов MS-DOS связывает их в цепочку, так, чтобы каждый драйвер содержал ссылку на ранее загруженный драйвер. Цепочка драйверов начинается, таким образом, с последнего загруженного драйвера (NUL-драйвер) и заканчивается самым первым загруженным драйвером. Такая цепочка строится, используя первые два слова заголовка каждого драйвера. Эти два слова содержат сегмент и смещение следующего в цепочке драйвера или, в случае последнего драйвера число – 1 (шестнадцатичное значение FFFF). Пример цепочки драйверов показан ниже.

Device	Type	Units	Attrib	Address	STRAT	INTRP
NUL	Char	01	8004	0000:1898	1418	141E
CON	Char	01	8013	08A9:0000	00A2	00AD
-----	Block	02	0000	083D:0000	00A7	00B2
CON	Char	01	8013	0070:0160	00A7	00B2
AUX	Char	01	8000	0070:01F1	00A7	00B8
PRN	Char	01	A000	0070:02A0	00A7	00C7
CLOCK\$	Char	01	8008	0070:034A	00A7	00DC
-----	Block	03	0800	0870:0416	00A7	00E2
COM1	Char	01	8000	0070:0203	00A7	00B8
LPT1	Char	01	A000	0070:02B2	00A7	00C7
LPT2	Char	01	A000	0070:0B13	00A7	00CD
LPT3	Char	01	A000	0070:0B25	00A7	00D3
COM2	Char	01	8000	0070:0B37	00A7	00BE
<<< ----- End Of Driver List ----- >>>						

Когда ДОС требуется обратиться к определенному драйверу, она начинает поиск по цепочке драйверов (начиная с NUL-драйвера) в порядке, обратном тому, в котором драйверы были загружены. После того как требуемый драйвер найден, ДОС обращается к нему с соответствующей командой. Последовательность поиска в цепочке при этом такова, что если загружен пользовательский драйвер, имя которого совпадает с именем какого-либо стандартного драйвера (такого, как CON, AUX или PRN), драйвер пользователя будет найден первым. Это позволяет пользователю заменять стандартные драйверы (например, заменить стандартный CON-драйвер на ANSI.SYS CON-драйвер). Стандартные драйверы в действительности загружаются и инициализируются до того, как файл CONFIG.SYS будет считан и обработан. Это позволяет процедуре инициализации драйвера использовать некоторые функции ДОС для вывода сообщений или настройке драйвера на конкретную версию операционной системы.

Далее приведем несколько конкретных примеров стандартных встроенных драйверов ДОО.

ДОО содержит встроенные драйверы для всех обслуживаемых им устройств. Кроме того, имеется ряд загружаемых драйверов, которые могут быть подключены к системе посредством команды DEVICE= в файле CONFIG.SYS.

Драйвер ANSI.SYS предназначен для:

- управления цветом экрана во время работы ДОО;
- позиционирования курсора в прикладных программах;
- переопределения клавиатуры.

Подключение:

DEVICE=ANSI.SYS.

Драйвер ANSI.SYS обеспечивает управление экраном дисплея и клавиатурой путем посылки последовательностей символов, начинающихся со специального символа – ESC (символ с кодом 27 или 1Bh). Подобные последовательности называются ESC-последовательностями. Символ ESC служит указателем для драйвера ANSI о том, что следующие за ним символы являются управляющими символами, а не обычным текстом. Послать текстовую строку на дисплей можно командами COPY, TYPE, ECHO. Назначение ESC-последовательностей соответствует стандартам Американского института национальных стандартов (ANSI) – отсюда название драйвера.

Драйвер DISLPAY.SYS предназначен для переключения таблицы шрифтов для дисплеев.

Подключение:

DEVICE=DISPLAY.SYS CON[:]=(Адаптер[,код] [,max[,N]]),

где:

Адаптер – тип дисплея, если этот параметр опущен, драйвер пытается сам определить тип подключенного к ПЭВМ дисплея;
код – аппаратная кодовая страница, т. е. шрифт, “прошитый” в знакогенераторе дисплея;
max – наибольшее количество шрифтов, которые могут быть аппаратно загружены в устройство;
N – указание, сколько шрифтов имеется для каждой кодовой страницы).

Драйвер PRINTER.SYS предназначен для переключения таблицы шрифтов на печатающем устройстве.

Подключение:

DEVICE = PRINTER.SYS LPTn=(ПУ,N,M),

где:

LPTn – номер устройства: LPT1, LPT2 или LPT3;

ПУ – тип печатающего устройства;

N – аппаратная кодовая страница, т. е. шрифт, “прошитый” в знакогенераторе принтера;

M – количество страниц, которое может быть загружено в устройство печати.

Драйвер VDISK.SYS предназначен для выделения части памяти под создание “виртуального диска”, на котором можно временно хранить файлы, к которым происходит частое обращение. Используется:

для ускорения доступа к определенным файлам;

для того, чтобы расширенная память не пропадала зря.

Подключение:

DEVICE = [d:][путь]VDISK.SYS [кбайт] [сект] [кат] [/E:n],

где:

d: – имя диска, на котором находится файл VDISK.SYS;

путь – имя каталога, где хранится этот файл;

кбайт – размер диска в килобайтах;

сект – размер сектора на диске (может быть 128, 256, 512 байт);

кат – максимальное число файлов в корневом каталоге;

/E:n – указание, что виртуальный диск должен располагаться в расширенной памяти (свыше 1M), n задает максимальное количество одновременно передаваемых секторов.

Виртуальный диск получит имя, следующее в алфавитном порядке за именами существующих дисков. Можно установить несколько виртуальных дисков. Например:

DEVICE = VDISK.SYS 30 – создание виртуального диска емкостью 30K;

DEVICE = VDISK.SYS 200 256 – создание виртуального диска емкостью 200K и с размером сектора 256 байт.

Драйвер SMARTDRV.SYS предназначен для кэширования программ – выделения специальной области памяти (кэш), играющей роль промежуточного буфера. Кэширование существенно повышает быстродействие за счет уменьшения времени обмена с кэшем и уменьшения числа обращений к диску. Область для кэширования располагается в расширенной или дополнительной памяти.

Подключение:

DEVICE = [d:][путь]SMARTDRV.SYS [разм] [/A],

где:

разм – размер памяти, используемый драйвером;

/A – указание на то, что для кэширования используется дополнительная память. Применяется в тех случаях, когда в распоряжении есть как расширенная, так и дополнительная память. Например:

device = smartdrv.sys 1024 /A – выделение 1024K дополнительной памяти;

device = smartdrv.sys /a – выделение всей дополнительной памяти.

2.1.7. Операционная система UNIX

Многие специалисты считают, что наибольшее влияние на развитие современных сетевых операционных систем оказали решения, использованные в ОС UNIX. Системы UNIX или типа UNIX работают на любых машинах. Доступность больших объемов оперативной памяти и мощных микропроцессоров привела к возрастанию интереса к многозадачности, системам мультипроцессорирования – сфере, в которой UNIX имеет отличную репутацию.

Системная среда не просто включена в систему UNIX – она и есть система UNIX. Вся система – UNIX, Си, команды, файлы и т.д. – это просто логический подход к функционированию компьютера.

Среда UNIX – это сочетание двух важнейших вещей: файлового дерева и интерфейса системных вызовов. Это дерево допускает бесконечное расширение возможностей, позволяя монтировать внешние дисковые области в любой точке файловой системы. Дерево помогает также в сборе логически связанных файлов, что делает систему более организованной.

Интерфейс системных вызовов обеспечивает набор инструментов, из которых можно построить большинство других функций.

Каждая компьютерная система поддерживает много различных сред. Эти среды используются как строительные блоки для создания функциональных рабочих систем. Различные уровни необходимы как для сокращения объема работы по управлению машиной, так и для построения такого интерфейса, чтобы мы могли использовать компьютер на относительно высоком, удобном для человека уровне. Такая модель помогает выстроить в ряд уровни, на которых мы можем работать. Имея больше знаний о том, где мы находимся в системе, и о том, как она функционирует вокруг нас, мы можем легче строить растущие абстрактные модели на вершине тех моделей, которые уже имеются.

Компьютеры – это фактически рабочие модели абстракций, представленные на рис. 2, который демонстрирует различные уровни, функционирующие внутри компьютера. Нижний слой – это стартовая точка, от которой многообразие растет вверх. Каждый уровень строится на предыдущем и используется для поддержки уровня, расположенного над ним. Для каждого более высокого уровня среда более объемна и более “виртуальна” в том смысле, что имеет место меньше условных ограничений. Верхние уровни используют для своей работы нижние и, таким образом, скрывают подробности, необходимые для работы этих нижних уровней. Можно создать модели высокого уровня, которые работают на машине более низкого уровня, не зная ничего о нижних уровнях.

Рассмотрим более подробно уровни модели.

На самом нижнем уровне находятся аппаратные средства и логические схемы. Этот уровень определяет способ хранения и



Рис. 2. Схема иерархической модели программно-аппаратных сред ЭВМ

обработки данных во всех аппаратных средствах. Поскольку технология изготовления микросхем продолжает развиваться, этот уровень становится физически меньше и проще, тогда как скорости запоминания и обработки продолжают расти. На этом уровне компонентами являются *центральный процессор (ЦП), память, микросхемы поддержки и системная шина*. Хотя прогресс на этом уровне продолжается, это вызывает очень малые изменения на верхнем слое пирамиды. Философия системы UNIX состоит в том, чтобы изолировать низкоуровневый аппаратный слой и обеспечить единообразные

интерфейсы к нему, которые не нуждаются в изменениях “наверху”. Верхний слой даже не должен знать о нижнем слое. Это не значит, что события в мире аппаратуры не важны в реальном мире, ведь противоречия реального мира влияют на скорость и емкость ресурсов, не говоря уже об их стоимости.

Уровень микропрограммной среды во многом похож на язык программирования. Он является инструментом, который использует архитектор системы для создания машинного языка. Машинный язык сообщает аппаратуре, какую конкретную команду следует выполнить. В начале эволюции ЭВМ большинство наборов команд были аппаратно кодированными. Это значит, что когда ЦП получал команду, декодирование и выполнение производилось непосредственно цепями в микросхеме. Благодаря прогрессу в технологии ЦП, некоторые микросхемы могут быть программируемыми на уровне исполнения команд, что позволяет конструкторам создавать и реализовывать новые наборы команд с минимальными усилиями.

Уровень виртуальной машины обеспечивает трансляцию из мнемонических команд языка ассемблера в коды операций и данные машинного языка. Язык ассемблера – это некоторая нотация, которая облегчает человеку понимание и управление работой компьютеров. Условная машина поддерживается ассемблером. Ассемблер может превращать идеи более высокого уровня в цепочки чисел (команды уровня процессора), которые затем выполняются. Наряду с ассемблером, применяются модели, помогающие использовать аппаратуру прерываний. Здесь определяются такие вещи, как стеки, вектора прерываний и периферийный ввод-вывод.

Ядро является следующим логическим продвижением вверх и концепцией, которую можно теперь реализовать программно на условной машине. Ядро предоставляет среду, поддерживающую еще большие абстракции, чем те, что рассматривались до сих пор. Двумя наиболее важными абстракциями на уровне ядра являются *управление процессами* для мультипрограммирования и многозадачности и *файловая система*, которая управляет хранением, форматом, поиском файлов и т.п. Когда эти две области интегрируются, мы имеем базовую функцию многопользовательской машины и ядро операционной системы.

Одной из наиболее важных областей, которыми управляет ядро, является безопасность. Проверки идентификации пользователя выполняются в системных вызовах внутри ядра. Определенные механизмы используются ядром для управления безопасностью файлов, устройств, памяти и процессов. Единственный способ отключить механизмы безопасности состоит в изменении исходного кода ядра и перекомпиляции всей системы.

Уровень ОС строится на ядре, чтобы создать полную операционную среду. Потребность в дополнительных функциях системы можно удовлетворить созданием автономных программ, имеющих конкретное

назначение. Таким образом, совокупность всех специфических функций определяет операционную систему.

Компилятор – это инструмент (или программа), построенный на операционной системе для дальнейшей разработки более совершенных и более мощных сред. Новые среды могут предполагать еще большие абстракции, чем на нижнем уровне, и делать больше допущений о том, что уже существует. Это делает возможным символические конструкции более высокого уровня, такие, как структуры данных и управляющие структур, из которых формируется прикладная программа.

С помощью компилятора можно определить совершенно новый язык и сделать его рабочим на компьютере, написав компилирующую программу, которая читает этот новый язык. Это открывает новые области во взаимодействии человека с машиной. Высокоуровневые языки могут воплощать различные подходы к решению задач, например, процедурную модель или объектно-ориентированную модель.

Программа, которая сделана с помощью компилятора, является прикладной программой. Примерами возможных прикладных программ является следующее поколение языков, интерпретаторов и генераторов прикладных программ. Интерпретатор – это программа, написанная на распространенном языке высокого уровня, которая может декодировать и исполнять исходный текст (в системе UNIX – командный процессор shell). Это программа на языке Си, созданная для чтения и исполнения команд, записанных по правилам синтаксиса, определенных командным процессором shell.

Генератор прикладных программ – это программа, написанная на языке высокого уровня. Она предназначена для получения достаточной информации от пользователя о его приложении и может использовать компиляторный язык, например Си, для написания прикладной программы, реализующей то, что требуется. Пользователь ничего не программирует. Выходом генератора является рабочая программа.

UNIX не делает особых различий между уровнями, но она вносит единообразие в этот широкий диапазон функций.

Верхний уровень – UNIX содержит средства для построения интерфейсов любого требуемого уровня сложности. Поскольку UNIX создавалась как многопользовательская система, многое сделано для того, чтобы система была безопасной и удобной для каждого пользователя. Каждому пользователю выделяется определенная часть файловой системы, которая полностью находится в его распоряжении. Он может заблокировать эту область так, чтобы никто не мог иметь доступ вовнутрь, или же может оставить ее открытой, чтобы другие пользователи могли читать эту область или писать в нее.

Регистрационный каталог – это не только область файловой памяти, но и вся пользовательская среда.

Решение, которое использует UNIX для создания файловой системы, – перевернутая модель дерева. Корень системы находится наверху, а

ветви растут в стороны и вниз. Имеется один и только один корень наверху. Ветви могут исходить в любом направлении и простираются вниз на любую глубину. Кроме того, можно иметь присоединяемые ветви, которые изымают из системы, а затем возвращают обратно. Они монтируются на существующую в системе древовидную структуру.

Когда пользователь регистрируется в системе, регистрационный каталог определяется в файле паролей. Создается персональная древовидная структура под соответствующим именем каталога. Она полностью принадлежит пользователю и может быть сделана недоступной для кого угодно, кроме корня (администратор-суперпользователь).

Как только регистрационный каталог X присоединен к определенному месту дерева, X получает полное управление структурой, которая существует ниже этого места. X может оставить ее плоской или сделать подобной дереву. Эта структура зависит фактически от его потребностей и развития предметной области пользователя. Древовидная структура регистрационного каталога представляет собой каркас среды, который может быть заполнен соответствующей информацией.

Операционная система, особенно такая высокоразвитая, как UNIX, имеет множество утилит и характерных особенностей. Сердцем UNIX является ядро (kernel). Ядро управляет процессами и руководит выполняемой работой. Оно также является своего рода мостом между аппаратурой и внешним миром. При работе с машиной много времени тратится на передачу данных, а это значит, что необходимо иметь дело со множеством различных типов устройств, каждое из которых имеет свои особенности.

UNIX был разработан так, чтобы облегчить управление данными и устройствами настолько, насколько это возможно. Со стороны ядра обращение ко всем внешним периферийным устройствам выполняется как к файлам устройств. Каждый тип устройств имеет свой собственный драйвер и специфическую архитектуру, но обращение к каждому устройству выполняется одинаковыми методами.

UNIX обращается к периферийным устройствам через “псевдофайлы”. Имеется два типа файлов – блочные и символьные. Оба типа имеют свое предназначение и особенности. Блочный – использует буферизацию и позволяет получить доступ к большим объемам данных на жестком диске. Символьный выполняет обмен с устройством по одному символу за обращение. Для этих файлов поддерживается все тот же механизм защиты, что и для всех других файлов в системе.

В заключение необходимо сказать, что система UNIX была полностью написана на языке Си.

2.2. Управление системными ресурсами

Память ЭВМ является одним из самых важных системных ресурсов. Рассмотрим вопросы управления памятью, формат памяти ДОС и

модели распределения ограниченной памяти для всех конкурирующих между собой целей.

Первоначально ДОС была разработана для устройств центрального процессора (CPU) 8086/8088, позволяющих адресовать суммарную память объемом 1 Мбайт. Типичное использование и размещение этой памяти было следующим:

1) Первые 10 сегментов (“кусков” по 64 кбайт) этой памяти относятся к области пользователя – область, объемом в 640Кбайт, в которой расположены сама ДОС и прикладные программы пользователя.

2) Оставшиеся 6 сегментов, составляющие в сумме 384 Кбайт, называются системной областью и резервируются для использования BIOS (базовая система ввода-вывода) и для связи с другими платами в системе. Это обобщенная модель, так как в действительности имеется много типов плат, которые используют эту область для многих целей, однако, будем рассматривать только общую схему.

С развитием аппаратных средств пределы адресуемой памяти расширялись, но общие принципы управления оставались в силе. Появилось понятие “расширенная память”, поскольку она простирается выше границы в 1 Мбайт и расширяет основной предел ДОС в 640 Кбайт.

Для ДОС ранних версий продукты расширяемой памяти доступны в трех разновидностях.

Первая спецификация расширяемой памяти называлась LIM EMS (Limit Expanded Memory Specification – предельная спецификация расширяемой памяти).

Несколько позже был разработан улучшенный стандарт AQA EEMS (the Enhanced Expanded Memory Specification – улучшенная спецификация расширяемой памяти).

Затем лучшие стороны этих стандартов были соединены в LIM EMS версии 4.0.

Все системы EMS состояли из памяти (на соединительной плате или плате расширения) и администратора улучшенной памяти EMM (the Enhanced Memory Manager) – устанавливаемого драйвера устройства. Для установки функций EMS резервируется прерывание ДОС с номером 67h. Поздние версии ДОС поддерживают стандарт LIM EMS версии 4.0 как часть операционной системы. Реализация аппаратных средств от производителя к производителю менялась.

Расширяемая память является результатом появления в среде ДОС устойчивых традиций использования страничной памяти или памяти коммутации банков. При этом подходе большой раздел памяти, который лежит вне адресного пространства процессора, “отображается” малыми областями на многие маленькие разделы памяти, лежащие внутри адресного пространства процессора. В то время как процессор не может адресовать большой раздел памяти непосредственно, он может выбрать или дойти до любой конкретной части, подобно выбору страницы в книге.

В спецификации расширяемой памяти EMS большая физическая

память отображается в 16-килобайтные разделы памяти ДОС, называемые страницами. Соответствующее 16-килобайтное адресное пространство в памяти называется *страничным фреймом*. Количество поддерживаемых страничных фреймов и размещение их внутри системы ДОС изменяется в зависимости от типа платы используемой расширяемой памяти, и существующей конфигурации системы.

Схема управления расширенной памятью работает следующим образом:

1. На ПЭВМ может быть установлена дополнительная память на одной или нескольких платах. В отличие от многообразных плат памяти накопитель на этих платах делится на страницы по 16 Кбайт. Расширенная память не адресуется приложениями DOS непосредственно, так как она не появляется в младших 640К адресного пространства персонального компьютера.

2. Эти платы памяти также содержат набор регистров соответствия, которые управляются программно для установления отображения какой-либо из 16-Кбайтных страниц на плате (платах) расширенной памяти на любую из четырех 16-Кбайтных зон в 64-Кбайтной части адресного пространства персонального компьютера, именуемого страничным кадром. Страничный кадр размещается где-то в резервном адресном пространстве персонального компьютера над 640К и ниже предела адресации (1М). Каждая зона в страничном кадре называется физической страницей и определяется числом 0 – 3. Страничный кадр образует окно, через которое правильно написанная программа может получить доступ ко всей емкости памяти платы (плат) расширенной памяти. Процесс изменения содержимого регистров отображения плат для обеспечения доступности страницы расширенной памяти программе называется страничным отображением.

3. Управление системой расширенной памяти, включая страничное отображение, выполняется программной компонентой, называемой Менеджер расширенной памяти или EMM (Expanded Memory Manager), которая поставляется изготовителем платы расширенной памяти. Во многом так же, как DOS и BIOS обеспечивают программный интерфейс между приложением и аппаратурой ПЭВМ, находящейся ниже него, менеджер расширенной памяти обеспечивает программный интерфейс между приложением и системой расширенной памяти. Менеджер расширенной памяти загружается в память так же, как драйвер клавиатуры DOS во время загрузки и сообщается с программами через программное прерывание 67h, используя механизм передачи параметров.

4. По запросу программы менеджер расширенной памяти размещает набор из одной или более логических страниц для программы. Он также размещает обработчик, который программа использует в последующих запросах расширенной памяти к менеджеру расширенной памяти, для определения набора страниц расширенной памяти, с которыми нужно работать. Во многом таким же образом как обработчик файлов DOS

используется для отслеживания файлов, открытых каждой программой, обработчики расширенной памяти используются менеджером расширенной памяти для отслеживания множества активных страниц расширенной памяти каждой программы. Номера логических страниц, связанные с обработчиком, отсчитываются относительно нуля до значения, на единицу меньшего, чем количество страниц, запрошенных программой.

5. Когда от менеджера расширенной памяти запрашивается обслуживание, программа определяет конкретную страницу в 16K расширенной памяти, которую она желает использовать, путем задания комбинации обработчика и номера логической страницы.

2.3. Управление вводом-выводом

Ввод-вывод – обмен данными под управлением ЭВМ – мы рассмотрим на примере работы последовательного порта.

Основной задачей последовательного порта является направление и получение данных по шине в виде потока битов (в противоположность параллельному порту, в котором внутренний байт передается целиком). Через последовательный порт можно подключать к системе “мышь” или модем для установления автоматической телефонной связи с сетевыми ресурсами.

При передаче данных, с точки зрения представления символов, необходимо передавать байты данных от одного устройства к другому, например, от персонального компьютера к модему. Если имеется восемь линий между двумя устройствами, то можно назначить каждой линии бит и послать сразу один байт данных. Это будет параллельная передача. Таким образом работает параллельный порт персонального компьютера, кроме того, в дополнение к восьми линиям данных имеются другие сигнальные линии, оказывающие помощь в передаче данных.

При наличии одной линии, необходимо посылать каждый байт данных последовательно, по одному биту. Данные могут посылаться синхронно, таким образом, что каждый байт посылается в определенное время (скажем, один байт каждые x секунд), или асинхронно со скоростью, которую предварительно определять необязательно. Последовательная связь дешевле, чем параллельная, так как требует меньше линий передачи данных – минимум две для двусторонней связи. Режим асинхронной передачи оказывает значительно меньшее воздействие на аппаратуру ввиду того, что не требуется дополнительное специальное оборудование для поддержки синхронизации между передатчиком и приемником. Таким образом, асинхронная последовательная связь является предпочтительным решением ввиду низкой стоимости и простоты используемых аппаратных средств. Конечно, в этом режиме необходимо преобразовывать каждый байт данных в серию битов и указывать приемнику начало и конец каждого байта.

Аппаратура последовательного порта в системах ДОС известна как последовательный адаптер или асинхронный связной адаптер (далее мы будем называть его последовательным адаптером). Адаптер основан на микросхеме (универсальный асинхронный приемопередатчик), имеет порт для подключения к модему и программируется посредством набора регистров. Микропроцессор имеет доступ к регистрам через ранее определенные адреса порта ввода-вывода.

Универсальный асинхронный приемопередатчик управляется посредством записи в набор регистров и чтения из них. Эти регистры доступны программисту через адреса порта. Адреса портов задаются последовательно, поэтому достаточно знать адрес первого порта. Он также известен как базовый адрес последовательного адаптера. В персональном компьютере двум последовательным портам COM1 и COM2 присвоены базовые адреса порта.

Существует два общих метода ввода-вывода в любой вычислительной системе: упорядоченный и управляемый прерываниями. Упорядоченность относится к повторяющейся проверке состояния регистра устройства ввода-вывода для инициализации требуемой транзакции. В упорядоченном вводе-выводе программа, запрашивающая символ ввода, многократно считывает состояние регистра в устройстве ввода-вывода до тех пор, пока оно не покажет, что символ доступен для ввода (или до тех пор, пока программа не решит, что “время закончилось”). Когда состояние указывает, что имеется готовый для работы символ, программа считывает его из соответствующего регистра устройства ввода-вывода. Сходная последовательность “ждать, до тех пор пока не готов, затем писать” используется при выведении символов на устройство ввода-вывода. Таким образом, дальнейшее выполнение программы приостанавливается до завершения выполнения операции ввода-вывода.

Большой проблемой для *упорядоченного* ввода-вывода через коммуникационный порт является то, что при определенной скорости передачи (выше 300 бод – 1 бод = 1 бит в секунду) программе трудно что-либо сделать с получаемым символом, кроме как отображать его на экране.

Рассмотрим следующий пример. Предположим, что мы читаем символы со скоростью 300 бод и имеем следующие связанные параметры: длина слова 7 бит, проверка на четность и один стоповый бит, который вместе со стартовым битом, добавляет до 10 бит на символ. Вы ожидаете получать около 30 символов каждую секунду. После чтения символа программа имеет около 1/30 секунды для выполнения других операций. Если Вы не желаете потерять какие-либо символы, то в это время Вы должны снова начать упорядочение ввода-вывода. Что произойдет, когда скорость возрастет до 9600 бод? Временной интервал между символами слишком мал для выведения символа на экран дисплея, не позволяет интерпретировать специальные символы и эмулировать (моделировать) терминал.

В подходе, основанном на *управлении прерываниями*, программа предоставляет возможность прерываниям устройства ввода/вывода поступать непосредственно на центральный процессор, который продолжает выполнять свою работу, не связываясь с устройством. Когда устройство готово к вводу-выводу, оно сигнализирует об этом центральному процессору. Получив сигнал, центральный процессор сохраняет свое текущее состояние и вызывает подпрограмму обслуживания прерываний, адрес которой хранится в таблице векторов прерываний. Эта подпрограмма выполняет операцию ввода-вывода, затем восстанавливает состояние машины и возвращает управление в прерванную программу.

Для регистра символов, поступающих в коммуникационный порт персонального компьютера, организовывается буфер. **Ввод с буферизацией** – способ организации взаимодействия с устройством ввода данных, при котором внешнее устройство независимо от программы выдает данные, а программа помещает их в буфер ввода до фактической обработки.

С помощью простой подпрограммы обработки прерываний, которая быстро считывает символ из коммуникационного порта и сохраняет его в следующей доступной ячейке памяти в буфере, символы не будут потеряны в процессе считывания и сохранения символа драйвером прерываний перед поступлением следующего символа. Эта задача достаточно проста для выполнения в короткие интервалы между поступающими символами. Время обработки главной программой символов, хранящихся в буфере, не имеет значения. Конечно, существует риск переполнения буфера, но эта проблема может быть решена простым увеличением его размера.

Для управления вводом-выводом символьной информации в ОС используются кодовые страницы

В MS-DOS предусмотрена возможность использования символов национальных алфавитов. Технически она основана на следующих особенностях ввода и обработки символов MS-DOS. Внутреннее представление текстовых символов задается кодом ASCII (American Standard Code for Information Interchange).

Всего в таблицу входит 256 символов, из них первые 128 содержат служебные символы и символы международного алфавита, следующие 128 отводятся под национальные символы разных стран. Каждой кодовой таблице, отличающейся второй половиной символов ASCII, присваивается уникальный номер, определяющий страну или язык, к которым эти символы относятся.

Отображение символов национального алфавита на экране становится возможным благодаря тому, что каждому шестнадцатичному коду можно поставить в соответствие матрицу пиксел, в которой содержится изображение символа. Разворачивание кода ASCII в матрицу стандартно осуществляет драйвер DISPLAY.SYS. Например,

матрица 8x8 для латинской буквы А имеет вид:

```
00000000
00111110
01000010
01000010
01111110
01000010
01000010
00000000
```

Аналогичным образом кодируются символы печатающего устройства. Все матрицы в упакованном виде хранятся в стандартных файлах с расширением .cpi (Code Page Information).

Следующей задачей MS-DOS является поддержка клавиатуры при работе с символами национального языка. Дело в том, что клавиатура вырабатывает не код символа ASCII, а просто порядковый номер нажатой клавиши (скан-код или код сканирования). Работа драйвера клавиатуры (KEYB) заключается в замене скан-кода клавиши кодом ASCII с помощью специальной таблицы соответствий. Такой принцип работы обеспечивает большую гибкость системы, так как можно менять расположение символов на клавиатуре.

Для работы с кодовыми страницами в MS-DOS имеются команды NLSFUNC, CHCP, KEYBxx.

Команда NLSFUNC предназначена для поддержки национальных форматов даты и времени. Синтаксис: NLSFUNC [[d:]имя_файла], где: "имя_файла" – имя файла, в котором хранится информация, характеризующая страну, если этот параметр опущен, информация будет взята из команды COUNTRY файла CONFIG.SYS (обычно национальная информация берется из файла COUNTRY.SYS, входящего в дистрибутивный набор ДОС).

Команда CHCP осуществляет выбор кодовой страницы или высвечивание текущей страницы на экране. Синтаксис: CHCP [код_табл], где код_табл – трехзначное число, указывающее номер кодовой страницы.

Команда KEYBxx подключает драйвер клавиатуры для закрепления за клавишами символов иностранного алфавита.

2.4. Управление внешней памятью и файловые системы

Введем ряд понятий, необходимых для дальнейшего изложения:

внешняя память – память, данные в которой доступны центральному процессору посредством операции ввода и вывода;

внешнее запоминающее устройство – запоминающее устройство, подключаемое к центральной части вычислительной системы и предназначенное для хранения большого объема данных;

файл изменений – файл, допускающий текущие добавления, подстановки и другие изменения его содержимого, которые могут затем вноситься в основной файл;

файловая система – часть операционной системы, обеспечивающая выполнение операций над файлами.

По мере того, как развивались операционные системы, обязанности по использованию и управлению системными ресурсами переходили от прикладных программ к системным компонентам, создававшимся для облегчения работы программиста. Это особенно видно на примере процессов управления хранением и доступом к данным.

На заре развития вычислительной техники каждая прикладная программа, которая обеспечивала хранение и доступ к данным на магнитной ленте или диске, должна была иметь свою собственную логику для обработки обращения к устройству. Программист должен был знать характеристики устройства и интерфейсов и затрачивать значительные усилия на управление распределением памяти на устройстве. При использовании устройства другими программами определение соглашений по использованию доставляло много неудобств, и трудно было избежать конфликтов.

Чтобы упростить эти действия, операционные системы обеспечили набор средств методов доступа. Задачи по использованию, управлению и совместному доступу к устройствам вошли в обязанности системы, что позволило прикладному программисту сосредоточить усилия на требованиях прикладных задач. Как только стали общедоступны языки программирования высокого уровня, непосредственное использование системных методов доступа было заменено соответствующими операторами языка. В результате был выработан действительно высокоуровневый подход к управлению данными для прикладных программистов, устранивший многие из трудностей (таких как соглашения об интерфейсе, обработка ошибок и управление буферами), которые оставались при использовании системных методов доступа.

Каждый язык высокого уровня развивал свои собственные файловые модели, но между языками происходило значительное взаимное обогащение, и постоянно развивался ряд совместно используемых концепций и понятий. Среди них:

- понятия записи;
- операции над последовательностью записей;
- методы доступа по ключу;
- файлы разделяемого доступа.

И так как развитие новых систем требовало реализации этих языков, методы доступа в них были построены так, чтобы поддерживать эти понятия и допускать разделяемый доступ к файлам для программ, написанных на разных языках.

При наличии этих достижений быстро стало очевидным, что прикладным системам необходим дополнительный файловый сервис. Так как число файлов возросло, стали необходимы системные средства по каталогизации и размещению файлов, наряду с системными средствами по обеспечению их целостности, копированию и дублированию для аварийного восстановления, для описаний их записей и для хранения перечня использующих их программ. Дополнительные системные средства также стали доступны в форме общих системных обслуживающий программ-утилит, служащих для ввода данных в файлы, для сортировки файлов, для копирования файлов и для генерации отчетов.

Эти тщательно разработанные наборы системных средств для управления, доступа и использования файлов в конце концов стали рассматриваться как *файловые системы* – организованные совокупности данных, обрабатываемых и управляемых всеобъемлющими и тесно связанными компонентами операционной системы. При дальнейшем расширении, эти файловые системы развивались навстречу требованиям прикладных разработок, связанных с использованием высокоуровневых языков. Эти файловые системы построены для поддержки файловых моделей нескольких языков программирования.

Для улучшения мобильности программ между системами файловые интерфейсы каждого из реализованных языков программирования стандартизованы. Файловые модели встроены в конструкции этих языков и могут различаться во многих случаях по целому ряду исторических причин.

Файловые системы ОС должны обеспечивать реализацию всех этих языковых средств, а файловые модели каждого языка программирования стандартизованы для обеспечения мобильности программ. В общем, файловые системы обеспечивают один или более из основных методов доступа, используемых компиляторами для реализации файловых моделей соответствующих им языков. Эти методы доступа могут рассматриваться как файловые модели нижнего уровня, каждая из которых обеспечивает набор необходимых функций, которые могут использоваться языками. Отображение языковой файловой модели на системную файловую модель не обязательно должно быть однозначным. В некоторых случаях – несколько запросов файловой системы могут быть использованы для поддержки одного языкового запроса. В других случаях запросы файловой системы должны быть параметризованы во избежание побочных эффектов, которых нет в языке.

При обработке в ОС распределенных файлов, архитектура управления распределенными данными определяет еще более низкий уровень файловых моделей. Чтобы получить доступ к удаленному файлу, системные файловые интерфейсы запрашивающей системы принимаются и преобразуются в последовательность команд файловой модели управления распределенными данными.

Общие программные интерфейсы к файлам, определенные каждым языком, отображаются соответствующим компилятором в локальный системный файловый интерфейс ОС. Для доступа к удаленным файлам эти интерфейсы затем отображаются в интерфейсы стандартной файловой системы управления распределенными данными. Отображение не обязательно должно быть однозначным, но оно обеспечивает высокую степень прозрачности для локальных и удаленных систем. Сложность выполнения преобразований интерфейсов определяется *логическим расстоянием* от языковых файловых моделей до файловых моделей управления распределенными данными. В общем, это не является проблемой благодаря стандартизации в проектировании как системных файловых моделей, так и файловых моделей управления распределенными данными.

Как правило, файловые системы поддерживают два различных вида файлов: потокоориентированные и записеориентированные файлы. *Потокоориентированные* файлы обеспечивают минимальный набор средств поддержки, необходимых для доступа к данным файла, но, тем не менее, предоставляют программисту максимум гибкости в организации и доступе к данным. При использовании этих файлов строки байт могут быть прочитаны или записаны по их относительному положению в файле.

В противоположность этому, записеориентированные файлы позволяют обеспечить широкий набор функций, используемых при реализации файловых моделей языков высокого уровня. Это реализует подход, согласно которому считается, что данные состоят из полей, организованных в записи, хранящихся в файлах.

Потокоориентированный файл предоставляет максимальную гибкость при хранении, доступе и управлении данными, поддерживая только наиболее простые функции. Но это не позволило избежать программных издержек; они только переместились из системной области в область прикладных программ, что значительно сократило возможности стандартизации и системной поддержки. Для языков высокого уровня каждый компилятор должен взять на себя обязанности по реализации в своих файловых моделях возможностей обработки потокоориентированных файлов.

Это не значит, что следует делать вывод о превосходстве потокоориентированного или записеориентированного подходов. При проектировании файловых систем делаются компромиссы между гибкостью и мощностью системного обслуживания, но каждая модель подходит для определенных типов приложений. Потокоориентированная модель больше соответствует сложно структурированным данным (таким как документы или выполняемые программы), в то время как записеориентированные модели более подходят для традиционной обработки коммерческих данных.

Термин *база данных* часто небрежно используется для обозначения развитой файловой системы, но эта неточная терминология нередко приводит к путанице. По мере того, как развивались и все более тщательно разрабатывались файловые системы, становились понятными общие концепции управления данными для обеспечения целостности, восстановления, секретности и гибкого доступа к данным, как и технология для их реализации. В результате стало ясно, что системы управления базами данных – это нечто качественно отличное от существующих файловых систем.

Для обеспечения этих требований возможности файловых систем могут быть расширены и в этих случаях системы управления базами данных (СУБД) развивались из файловой системы. Но в большинстве систем поддержка баз данных была добавлена отдельно.

Технология баз данных предоставляет много возможностей, имеющих большое значение для некоторых приложений. Однако, для многих приложений, также содержащих значительные объемы данных, просто нет необходимости в средствах баз данных. Примерами таких данных являются программы (как исходные, так и объектные модули), личные файлы, документы, изображения, телеконференции и другие формы данных без регулярной структуры, которая лежит в основе большинства конструкций баз данных. Для их хранения и обращения к ним более эффективно простое файловое обслуживание.

Файловые системы не вытеснены системами баз данных. Они остаются полезными и экономичными для многих приложений и будут поддерживаться операционными системами в ближайшем обозримом будущем. Поскольку операционные системы продолжают развиваться на пути к более всесторонним сервисным средствам, то их файловые системы должны развиваться вместе с ними. Сегодня это развитие идет к обслуживанию, распределенному по системам телекоммуникационных сетей.

В течение многих лет средства телекоммуникации используются для передачи данных между вычислительными системами. Эти данные в основном состоят из потоков данных, электронной почты, документов, целых файлов и сообщений по распределенным прикладным системам. Но теперь появилась необходимость в управлении файлами распределенными на различных узлах телекоммуникационных сетей ЭВМ. Необходимость в создании эффективных распределенных файловых систем определяется рядом обстоятельств.

Сети ЭВМ выросли в размерах и по сложности, при этом данные и пользователи часто оказывались в разных узлах сети. Доступ пользователей к необходимым им данным стал затруднен. Стоимость разрабатываемых специализированных коммуникационных программ для чтения и обновления файлов, хранящихся в удаленных системах, попросту слишком высока для большинства приложений и совершенно

неприемлема при нечастых или случайных обращениях. Распределенные файловые системы должны сводить эту стоимость по существу к стоимости использования средств коммуникации.

Необходимость актуализации данных в распределенных системах требует стандартизации ведения распределенных файловых систем на всех узлах.

Для обеспечения доступности данных совместное использование файлов является основным условием. Файлы, заблокированные или назначенные отдельными пользователями на большие периоды времени, недоступны остальным. Хорошо спроектированная файловая система старается максимизировать параллельность доступа пользователей к файлам.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

- 1. Составьте логическую схему базы знаний по теме юниты.*

2. Изобразите на схеме основные компоненты программного обеспечения разработчика (структуру ИСП/IDE).

3. Впишите в представленную таблицу содержание работ на каждом этапе загрузки ДОС после включения питания ПЭВМ.

Наименование этапа загрузки	Содержание работ на данном этапе

4. Составьте таблицу (схему) распределения областей памяти (ОЗУ) ПЭВМ в мегабайтовом адресуемом пространстве между компонентами ОС и прикладными программами.

5. Заполните представленную таблицу.

Название компоненты ОС/360	Содержание работ, выполняемых данной компонентой

6. Заполните представленную таблицу.

Вид прерывания ДОС	Содержание работ, выполняемых данным видом прерываний

7. Для управления выполнением программ в ДОО используется Program Segment Prefix (PSP). PSP - это управляющая область в 256 байт, которая строится в памяти в начале каждой программы. Она содержит различные поля, используемые ДОО для управления выполнением программы. Запишите в таблицу содержание шести из этих полей:

Номер поля PSP	Содержание поля PSP
1	
2	
3	
4	
5	
6	

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

ЮНИТА 1

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Редактор Н.В. Друх
Оператор компьютерной верстки Д.В. Федотов

Изд. лиц. ЛР № 071765 от 07.12.1998	Сдано в печать
НОУ "Современный Гуманитарный Институт"	
Тираж	Заказ
