2. Структура операционных систем

Достаточно условно все программные модули, входящие в состав ОС, можно разбить на две группы: базовое **ядро** и вспомогательные модули **(надстройка).** Линия раздела между этими группами различна в разных ОС. Модули ядра реализуют основные жизненно важные функции ОС, такие как **управление памятью, отработка прерываний, переключение процессов.**

Важной особенностью модулей ядра является то, что они выполняются в **привилегированном** режиме работы процессора (режим ядра, или **kernel mode**) и после запуска системы постоянно находятся в основной памяти. В привилегированном режиме разрешено выполнение **всех** команд процессора, в том числе и таких, выполнение которых запрещено в пользовательском режиме. Наиболее важными привилегированными командами являются команды низкоуровневого ввода и вывода байтов при взаимодействии с устройствами.

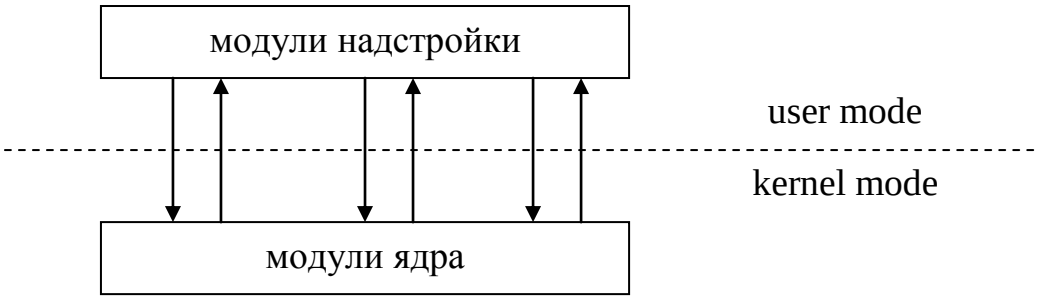
В отличие от этого, модули надстройки выполняются в **пользовательском** режиме (**user mode**), и с этой точки зрения они ничем не отличаются от обычных прикладных программ. В этом режиме попытка выполнения запрещенных команд приводит к немедленному прекращению работы программы. Поэтому для реализации своего поведения модули надстройки должны обращаться к стандартным системным функциям. Такая организация ОС имеет как преимущества, так и недостатки. Важнейшими преимуществами являются:

- более надежная работа системы за счет защиты модулей ядра от сбоев в работе модулей надстройки;
- простота внесения изменений и дополнений в состав надстройки за счет замены соответствующих файлов с последующим подключением на основе механизма динамической компоновки.

Основным недостатком является некоторое **замедление** выполнения модулей надстройки, т.к. практически каждый вызов системной функции требует переключения процессора в привилегированный режим, а потом –

обратно. Для ускорения этих операций они чаще всего выполняются на аппаратном уровне.

К модулям надстройки обычно относятся трансляторы с языков программирования, компоновщики, программы-отладчики, редакторы, вспомогательные утилиты.



В свою очередь, ядро системы может реализовываться разными способами. Наиболее простой и исторически самый первый способ – так называемое **монолитное** ядро. В этом случае все входящие в ядро подпрограммы полностью равноправны и могут вызывать друг друга для выполнения необходимых действий. Большим преимуществом монолитных ядер является высокая скорость работы, а большим недостатком – сложность внесения изменений в код ядра, требующая перекомпиляции и перекомпоновки подпрограмм ядра. Как следствие, сильно затрудняется перенос системы на другие аппаратные платформы.

Поэтому монолитное ядро пригодно для относительно несложных ОС, когда еще можно отследить связи подпрограмм друг с другом. В сложных ОС этих связей становится слишком много, и как всегда для борьбы с возрастающей сложностью можно провести структуризацию ядра, сгруппировав логически связанные подпрограммы по уровням. Подобная организация ядра иногда называется **многослойной** или **многоуровневой.**

Чем ниже расположен слой, тем ближе он к аппаратуре компьютера, чем выше – тем дальше от аппаратуры и, наоборот, ближе к прикладным программам. Тем самым обеспечивается относительная **независимость**

подпрограмм высших уровней от аппаратных особенностей процессорной платформы. Многослойную организацию ядра можно представить следующим образом:

ядро	интерфейс системных вызовов (API)
	высокоуровневые менеджеры-диспетчеры ресурсов
	базовые исполнительные модули
	аппаратно-зависимые модули

аппаратная поддержка функций ОС

Дадим краткую характеристику каждого из уровней, рассматривая их снизу вверх. Прежде всего, необходимо отметить, что все современные процессоры в той или иной степени на аппаратном уровне обеспечивают выполнение следующих функций, напрямую связанных с задачами ОС:

- поддержка привилегированного/пользовательского режимов работы процессора за счет использования специального одно-двухбитового признака режима, который устанавливается в соответствии с правами выполняемой программы и позволяет тем самым проверять каждую выполняемую команду; попытка выполнить запрещенную команду автоматически распознается с генерацией особой ситуации (исключения);
- аппаратное переключение процессора с выполнения одного потока команд на другой поток, что требует запоминания состояния прерываемой программы для последующего ее возобновления; для этого в специальной области основной памяти запоминаются значения всех основных регистров, включая счетчик команд, регистр состояния и системные регистры;
- средства преобразования виртуальных адресов в физические; поскольку эти преобразования выполняются очень часто, чрезвычайно важно реализовать их в максимально возможной степени именно на аппаратном уровне; для этого в составе процессора предусматриваются

специальные системные регистры; алгоритмы преобразования более подробно рассматриваются в темах 7-9;

- обработка прерываний как важнейшего механизма функционирования любой вычислительной системы; эта обработка включает распознавание момента возникновения и типа прерывания и быстрый переход на подпрограмму обработки этого прерывания;
- защита областей памяти с кодом и данными одной программы от несанкционированного вмешательства со стороны других программ.

Самый нижний уровень в многослойной организации ядра занимают программные модули, наиболее тесно связанные с базовой аппаратной платформой и поэтому называемые **аппаратно-зависимыми.** Наличие этого уровня объясняется стремлением достичь двух противоречивых целей – сделать ОС максимально быстрой (что возможно именно за счет учета особенностей конкретной платформы) и в то же время – максимально универсальной. Модули этого уровня позволяют в идеале полностью изолировать вышележащие модули от особенностей конкретной платформы. Это (опять же в идеале) позволяет переходить к другим платформам лишь за счет изменения модулей нижнего уровня, совсем (или почти совсем) не трогая модули более высоких уровней.

В качестве примера аппаратно-зависимого модуля можно привести (правда, с некоторыми оговорками) известную систему **BIOS** (Basic Input/Output System) для процессоров Intel. Эта система представляет собой программный код, реализующий простейшие низкоуровневые операции с основными устройствами ввода/вывода (клавиатура, мышь, монитор, диски), но на их основе можно строить более сложные подпрограммы.

Следующий уровень образуют «рабочие лошадки» – программные модули, реализующие все основные операции по переключению процессов, обработке прерываний, реализации страничной организации памяти, взаимодействию процессов и т.д. Эти модули с одной стороны используют

базовые механизмы нижнего уровня, а с другой – реализуют решения, принятые соответствующими менеджерами на более высоком уровне.

Набор управляющих подпрограмм (**менеджеров** или **диспетчеров** ресурсов) и составляет следующий более высокий уровень ядра. Стандартный набор таких подпрограмм включает: диспетчер процессов, диспетчер памяти, диспетчер ввода/вывода, диспетчер файловой системы. Алгоритмы работы этих диспетчеров рассматриваются в следующих темах пособия.

Наконец, самый верхний уровень ядра образуют **системные API-вызовы.** С точки зрения разработчика ПО эти вызовы оформлены как обычные функции, часто – с передачей входных параметров и возвратом результата отработки вызова. Число таких системных вызовов может быть весьма различным. Например, набор API-функций в системах семейства Windows (Win32 API) насчитывает до 2000 вызовов, реализующих практически все аспекты функционирования систем данного класса.

Рассмотренная многослойная организация ОС является лишь одной из возможных. Уменьшение числа слоев до одного-двух позволяет достичь большего быстродействия, что может быть очень важным для систем реального времени, но с другой стороны, снижает универсальность, переносимость и расширяемость таких систем.

В последнее время большую популярность получают системы с **микроядерной** организацией. Микроядро – это минимально необходимый набор подпрограмм, реализующих в привилегированном режиме лишь самые необходимые функции. В качестве микроядра можно взять модули уровней 1 и 2 на приведенной выше схеме. Модули уровней 3 и 4 выполняются в пользовательском режиме как обычные прикладные программы. Основой взаимодействия разноуровневых модулей является **механизм сообщений,** реализованный по технологии **«клиент – сервер»:**

- Клиент в лице приложения или системного диспетчера формирует запрос-сообщение на вызов системной функции или другого диспетчера (запрос серверу).
- Этот запрос с помощью диспетчера сообщений, работающего в составе микроядра, передается вызванному системному модулю (серверу) обратно на пользовательский уровень.
- После отработки системного запроса сервером его результат опять же через микроядро возвращается клиенту.

Из данной схемы видно, что в два раза увеличивается число переключений между режимами, что немного снижает скорость работы системы и является основным недостатком микроядерной архитектуры. Однако этот недостаток с лихвой компенсируется **достоинствами** микроядерной организации, среди которых можно отметить следующие:

- более **высокая надежность** работы, т.к. каждый сервер-диспетчер работает как самостоятельный процесс и тем самым надежно защищен от нежелательного вмешательства со стороны других серверов;
- **расширяемость** за счет добавления новых серверов с четко определенным интерфейсом с микроядром;
- **распределенность,** т.е. возможность запускать серверы и микроядро на физически различных компьютерах.

Микроядро обычно выполняет следующие функции: управление основной памятью, примитивное управление процессами, базовое управление вводом/выводом и прерываниями, передача сообщений. В качестве серверов реализуются диспетчеры процессов, модули управления внешними устройствами, подсистема управления файлами, программные прикладные интерфейсы.