Основные принципы построения операционных систем

Мазинов Рустик

Основные принципы построения операционных систем

Среди множества принципов построения операционных систем перечислим наиболее важных: принцип модульности, принцип виртуализации, мобильности (переносимости) и совместимости, принцип открытости, генерации операционной системы из программных компонентов и некоторые другие.

Принцип модульности

Операционная система строится из множества программных модулем в общем случае понимают функционально законченный элем выполненный в соответствии с принятыми межмодульными интерфейсами. По своему определению модуль предполагает легкий способ его замены другим при наличии заданных интерфейсов. Способы обособления составных частей операционной системы в отдельные модули могут быть существенно разными, но чаще всего разделение происходит именно по функциональному признаку. В значительной степени разделение системы на модули определяется используемым методом проектирования системы (снизу вверх или наоборот).

Особо важное значение при построении операционных систем имеют привилегированные, повторно входимые и реентерабельные модули, ибо они позволяют более эффективно использовать ресурсы вычислительной системы. Как мы уже знаем, свойство реентерабельности может быть достигнуто различными способами, но чаще всего используются механизмы динамического выделения памяти под переменные для нового вычислительного процесса (задачи). В некоторых системах реентерабельность программы получают автоматически. Этого можно достичь благодаря неизменяемости кодовых частей программ при исполнении, а также автоматическому распределению регистров, автоматическому отделению кодовых частей программ от данных и помещению последних в системную область памяти, которая распределяется по запросам от выполняющихся задач. Естественно, что для этого необходима соответствующая аппаратная поддержка. В других случаях это достигается программистами за счет использования специальных системных модулей.

Принцип модульности отражает технологические и эксплуатационные системы. Наибольший эффект от его использования достижим в случае, когда принцип распространен одновременно на операционную систему, прикладные программы и аппаратуру. Принцип модульности является одним из основных в UNIX системах.

Во всех операционных системах можно выделить некоторую часть наиболее важных управляющих модулей, которые должны постоянно находиться в оперативной памяти для более скорой реакции системы на возникающие события эффективной организации вычислительных процессов. Эти модули вместе с некоторыми системными структурами данных, необходимыми для функционирования операционной системы, образуют так называемое ядро операционной системы, так как это действительно ее самая главная, центральная часть, основа системы.

При формировании состава ядра требуется удовлетворить двум противоречивым требованиям. В состав ядра должны войти наиболее часто используемые системные модули. Количество модулей должно быть таким, чтобы объем памяти, занимаемый ядром, был не слишком большим. В его состав, как правило, входят модули по управлению системой прерываний, средства по переводу программ из состояния счета в состояние ожидания, готовности и обратно, средства по распределению основных ресурсов, таких как оперативная память и процессор.

Принцип особого режима работы

Ядро операционной системы и низкоуровневые драйверы, управляющие работой каналов и устройств ввода-вывода, должны работать в специальном режиме работы процессора. Это необходимо по нескольким причинам. Во-первых, введение специального режима работы процессора, в котором должен исполняться только код операционной системы, позволяет существенно повысить надежность выполнения вычислений. Это касается выполнения, как управляющих функций самой операционной системы, так и прикладных задач пользователей. Категорически нельзя допускать, чтобы какая-нибудь прикладная программа могла вмешиваться (преднамеренно или в связи с появлением ошибок вычислений) в вычисления, связанные с супервизорной частью операционной системы.

- Во-вторых, ряд функций должен выполняться исключительно централизованно, под управлением операционной системы. К этим функциям мы, прежде всего, должны отнести функции, связанные с управлением процессами ввода-вывода данных. Вспомните основные принципы организации ввода-вывода: все операции ввода-вывода данных объявляются привилегированными. Это легче всего сделать, если процессор может работать, как минимум, в двух режимах: привилегированном (режим супервизора) и пользовательском. В первом режиме процессор может выполнять все команды, тогда как в пользовательском набор разрешенных команд ограничен. Естественно, что помимо запрета на выполнение команд ввода-вывода в пользовательском режиме работы процессор не должен позволять обращаться к своим специальным системным регистрам — эти регистры должны быть доступны только в привилегированном режиме, то есть исключительно супервизорному коду самой операционной системы. Попытка выполнить запрещенную команду или обратиться к запрещенному регистру должна вызывать прерывание (исключение), и центральный процессор должен быть предоставлен супервизорной части операционной системы для управления выполняющимися вычислениями.

Поскольку любая программа требует операций ввода-вывода, прикладные программы для выполнения этих (и некоторых других) операций обращаются к супервизорной части операционной системы (модуль супервизора иногда называют супервизором задач) с соответствующим запросом. При этом процессор должен переключиться в привилегированный режим работы. Чтобы программы не могли произвольным образом обращаться к супервизорному коду, который работает в привилегированном режиме, им предоставляется возможность обращаться к нему в строгом соответствии с принятыми правилами. Каждый запрос имеет свой идентификатор и должен сопровождаться соответствующим количеством параметров, уточняющих запрашиваемую у операционной системы функцию (операцию). Поэтому супервизор задач при получении запроса сначала его тщательно проверяет. Если запрос корректный и программа имеет право с ним обращаться, то запрос на выполнение операции, как правило, передается соответствующему модулю операционной системы. Множество запросов к операционной системе образует соответствующий системный интерфейс прикладного программирования (Application Program Interface, API).

Принцип виртуализации

В наше время уже не требуется пояснять значение слова “виртуальный”, ибо о виртуальных мирах, о виртуальной реальности знают даже дети. Принцип виртуализации нынче используется практически в любой операционной системе. Виртуализация ресурсов позволяет не только организовать разделение тех ресурсов между вычислительными процессами, которые не должны разделяться. Виртуализация позволяет абстрагироваться от конкретных ресурсов, максимально обобщить их свойства и работать с некоторой абстракцией, вобравшей в себя наиболее значимые особенности. Этот принцип позволяет представить структуру системы в виде определенного набора планировщиков процессов и распределителей ресурсов (мониторов) и использовать единую централизованную схему распределения ресурсов.

Следует заметить, что сама операционная система существенно изменяет наши представления о компьютере. Она виртуализирует его, добавляя ему функциональности, удобства управления, предоставляя средства организации параллельных вычислений и т. д. Именно благодаря операционной системе мы воспринимаем компьютер совершенно иначе, чем без нее.

Наиболее законченным и естественным проявлением концепции виртуальности является понятие виртуальной машины. По сути, любая операционная система, являясь средством распределения ресурсов и организуя по определенным правилам управление процессами, скрывает от пользователя и его приложений реальные аппаратные и иные ресурсы, заменяя их некоторой абстракцией. В результате пользователи видят и используют виртуальную машину как некое устройство, способное воспринимать их программы, написанные на определенном языке программирования, выполнять их и выдавать результаты на виртуальные устройства, которые связаны с реально существующими в данной вычислительной системе. При таком языковом представлении пользователя совершенно не интересует реальная конфигурация вычислительной системы, способы эффективного использования ее компонентов и подсистем. Он мыслит и работает с машиной в терминах используемого им языка.

Чаше виртуальная машина, предоставляемая пользователю, воспроизводит архитектуру реальной машины, но архитектурные элементы в таком представлении выступают с новыми или улучшенными характеристиками, часто упрощающими работ}' с системой. Характеристики могут быть произвольными, но чаще всего пользователи желают иметь собственную “идеализированную” по архитектурным характеристикам машину в следующем составе.

Единообразная по логике работы память (виртуальная) достаточного для выполнения приложений объема. Организация работы с информацией в такой памяти производится в терминах работы с сегментами данных на уровне выбранного пользователем языка программирования.

Произвольное количество процессоров (виртуальных), способных работать параллельно и взаимодействовать во время работы. Способы управления процессорами, в том числе синхронизация и информационные взаимодействия,

реализованы и доступны пользователям с уровня используемого языка в терминах управления процессами.

Произвольное количество внешних устройств (виртуальных), способных работать с памятью виртуальной машины параллельно или последовательно, асинхронно или синхронно по отношению к работе того или иного виртуального процессора, которые инициируют работу этих устройств. Информация, передаваемая или хранимая на виртуальных устройствах, не ограничена допустимыми размерами. Доступ к такой информации осуществляется на основе либо последовательного, либо прямого способа доступа в терминах соответствующей системы управления файлами. Предусмотрено расширение информационных структур данных, хранимых на виртуальных устройствах.

Принцип мобильности

Мобильность, или переносимость, означает возможность и легкость переноса операционной системы на другую аппаратную платформу. Мобильная операционная система обычно разрабатывается с помощью специального языка высокого уровня, предназначенного для создания системного программного обеспечения. Такой язык помимо поддержки высокоуровневых операторов, типов данных и модульных конструкций должен позволять непосредственно использовать аппаратные возможности и особенности процессора. Кроме этого, такой язык должен быть широко распространенным и реализованным в виде систем программирования, которые либо уже имеются на целевой платформе, либо позволяют получать программные коды для целевого компьютера. Другими словами, этот язык системного программирования должен быть достаточно распространенным и технологичным. Одним из таких языков является язык С. В последние годы язык С++ также стал использоваться для этих целей, поскольку идеи объектно-ориентированного программирования оказались плодотворными не только для прикладного, но и для системного программирования. Большинство современных операционных систем были созданы именно как объектно-ориентированные.

Обеспечить переносимость операционной системы достаточно сложно. Дело в том, что архитектуры разных процессоров могут очень сильно различаться. У них может быть разное количество рабочих регистров, причем часть регистров может оказаться контекстно-зависимыми, как это имеет место в процессорах с архитектурой ia32. Различия могут быть и в реализации адресации. Более того, для операционной системы важной является не только архитектура центрального процессора, но и архитектура компьютера в целом, ибо важнейшую роль играет подсистема ввода-вывода, а она строится на дополнительных (по отношению к центральному процессору) аппаратных средствах.

В таких условиях сделать эффективным код операционной системы при условии создания его на языке типа C/C++ невозможно. Поэтому часть программных модулей, которые более всего зависят от аппаратных особенностей процессора, от типов поддерживаемых данных, способов адресации, системы команд и других важнейших моментов, разрабатывается на языке ассемблера. Очевидно, что модули, написанные на языке ассемблера, при переносе операционной системы на процессор с иной архитектурой должны быть написаны заново. Зато остальная (большая) часть кода операционной системы может быть просто перекомпилирована под целевой процессор. Именно по этому принципу в свое время была создана операционная система UNIX. Относительная легкость переноса этой системы на другие компьютеры позволила сделать ее одной из самых распространенных. Для обеспечения мобильности был даже создан стандарт на интерфейс прикладного программирования, названный POSIX (Portable Operating System Interface for Computer Environments — интерфейс прикладного программирования для переносимых операционных систем).

К сожалению, на самом деле далеко не все операционные системы семейства UNIX допускают относительно простую переносимость созданного для них программного обеспечения, хотя сами они и поддерживают такую переносимость. Основная причина тому — отход от единого стандарта API — POSIX. Очевидно, что платой за универсальность, прежде всего, является потеря производительности при выполнении операций ввода-вывода и вычислений, связанных с этими операциями. Поэтому ряд разработчиков шли и до сих пор идут на отказ от принципа мобильности, поскольку не всегда следование этому принципу экономически оправдано.

Спасибо за внимание