1.5. Критерии оценки ОС

При сравнительном рассмотрении различных ОС в целом или их отдельных подсистем возникает вечный вопрос — какая из них лучше и почему, какая архитектура системы предпочтительнее, какой из алгоритмов эффективнее, какая структура данных удобнее и т.п.

Очень редко можно дать однозначный ответ на подобные вопросы, если речь идет о практически используемых системах. Система или ее часть, которая хуже других систем во всех отношениях, просто не имела бы права на существование. На самом деле имеет место типичная многокритериальная задача: имеется несколько важных критериев качества, и система, опережающая прочие по одному критерию, обычно уступает по другому. Сравнительная важность критериев зависит от назначения системы и условий ее работы.

1.5.1. Надежность

Этот критерий вообще принято считать самым важным при оценке программного обеспечения, и в отношении ОС его действительно принимают во внимание в первую очередь.

Что понимается под надежностью ОС?

Прежде всего, ее **живучесть**, т.е. способность сохранять хотя бы минимальную работоспособность в условиях аппаратных сбоев и программных ошибок. Высокая живучесть особенно важна для ОС компьютеров, встроенных в аппаратуру, когда вмешательство человека затруднено, а отказ компьютерной системы может иметь тяжелые последствия.

Во-вторых, способность, как минимум, диагностировать, а как максимум, компенсировать хотя бы некоторые типы аппаратных сбоев. Для этого обычно вводится избыточность хранения наиболее важных данных системы.

В-третьих, ОС не должна содержать собственных (внутренних) ошибок. Это требование редко бывает выполнимо в полном объеме (программисты давно сумели доказать своим заказчикам, что в любой большой программе всегда есть ошибки, и это в порядке вещей), однако следует хотя бы добиться, чтобы основные, часто используемые или наиболее ответственные части ОС были свободны от ошибок.

Наконец, к надежности системы следует отнести ее способность противодействовать явно неразумным действиям пользователя. Обычный пользователь должен иметь доступ только к тем возможностям системы, которые необходимы для его работы. Если же пользователь, даже действуя в рамках своих полномочий, пытается сделать что-то очень странное (например, отформатировать системный диск), то самое малое, что должна

сделать ОС, это переспросить пользователя, уверен ли он в правильности своих действий.

1.5.2. Эффективность

Как известно, эффективность любой программы определяется двумя группами показателей, которые можно обобщенно назвать «время» и «память». При разработке системы приходится принимать много непростых решений, связанных с оптимальным балансом этих показателей.

Важнейшим показателем временно́й эффективности является **производительность** системы, т.е. усредненное количество полезной вычислительной работы, выполняемой в единицу времени. С другой стороны, для диалоговых ОС не менее важно **время реакции** системы на действия пользователя. Эти показатели могут в некоторой степени противоречить друг другу. Например, в системах разделения времени увеличение кванта времени повышает производительность (за счет сокращения числа переключений процессов), но ухудшает время реакции.

В программировании известна аксиома: выигрыш во времени достигается за счет проигрыша в памяти, и наоборот. Это в полной мере относится к ОС, разработчикам которых постоянно приходится искать баланс между затратами времени и памяти.

Забота от эффективности долгое время стояла не первом месте при разработке программного обеспечения, и особенно ОС. К сожалению, оборотной стороной стремительного увеличения мощности компьютеров стало ослабление интереса к эффективности программ. В настоящее время эффективность является первостепенным требованием разве что в отношении систем реального времени.

1.5.3. Удобство

Этот критерий наиболее субъективен. Можно предложить, например, такой подход: система или ее часть удобна, если она позволяет легко и просто решать те задачи, которые встречаются наиболее часто, но в то же время содержит средства для решения широкого круга менее стандартных задач (пусть даже эти средства не столь просты). Пример: такое частое действие, как копирование файла, должно выполняться при помощи одной простой команды или легкого движения мыши; в то же время для изменения разделов диска не грех почитать руководство, поскольку это может понадобиться даже не каждый год.

Разработчики каждой ОС имеют собственные представления об удобстве, и каждая ОС имеет своих приверженцев, считающих именно ее идеалом удобства.

1.5.4. Масштабируемость

Довольно странный термин «масштабируемость» (scalability) означает возможность настройки системы для использования в разных вариантах, в зависимости от мощности вычислительной системы, от набора конкретных периферийных устройств, от роли, которую играет конкретный компьютер (сервер, рабочая станция или изолированный компьютер) от назначения компьютера (домашний, офисный, исследовательский и т.п.).

Гарантией масштабируемости служит продуманная модульная структура системы, позволяющая в ходе установки системы собирать и настраивать нужную конфигурацию. Возможен и другой подход, когда под общим названием объединяются, по сути, разные системы, обеспечивающие в разумных пределах программную совместимость. Примером могут служить версии Windows NT/2000/XP, Windows 95/98 и Windows CE.

В некоторых случаях фирмы, производящие программное обеспечение, искусственно отключают в более дешевых версиях системы те возможности, которые на самом деле реализованы, но становятся доступны, только если пользователь покупает лицензию на более дорогую версию. Но это уже вопрос, связанный не с технической стороной дела, а с маркетинговой политикой.

1.5.5. Способность к развитию

Чтобы сложная программа имела шансы просуществовать долго, в нее изначально должны быть заложены возможности для будущего развития.

Одним из главных условий способности системы к развитию является хорошо продуманная модульная структура, в которой четко определены функции каждого модуля и его взаимосвязи с другими модулями. При этом создается возможность совершенствования отдельных модулей с минимальным риском вызвать нежелательные последствия для других частей системы.

Важным требованием к развитию ОС является **совместимость версий снизу вверх**, означающая возможность безболезненного перехода от старой версии к новой, без потери ранее наработанных прикладных программ и без необходимости резкой смены всех навыков пользователя. Обратная совместимость — сверху вниз — как правило, не гарантируется, поскольку в ходе развития система приобретает новые возможности, не реализованные в старых версиях. Программа из Windows 3.1 будет нормально работать и в Windows XP; наоборот — вряд ли.

Фирмы-производители ОС прилагают максимум усилий для обеспечения совместимости снизу вверх, чтобы не отпугнуть пользователей. Но при этом фирмы стараются в каждую новую версию заложить какую-нибудь новую конфетку, которая побудила бы пользователей как можно скорее купить ее.

Совместимость версий — благо для пользователя, однако на практике она часто приводит к консервации давно отживших свой век особенностей

или же просто неудачных решений, принятых в ранней версии системы. В документации подобные архаизмы помечаются как «устаревшие» (obsolete), но полного отказа от них, как правило, не происходит (а вдруг где-то еще работает прикладная программа, написанная двадцать лет назад с использованием именно этих средств?).

Как правило, наиболее консервативной стороной любой ОС являются не алгоритмы, а структуры системных данных, поэтому дальновидные разработчики заранее строят структуры «на вырост»: закладывают в них резервные поля, используют переменные вместо некоторых констант, устанавливают количественные ограничения с большим запасом и т.п.

1.5.6. Мобильность

Под **мобильностью** (portability) понимается возможность переноса программы (в данном случае ОС) на другую аппаратную платформу, т.е. на другой тип процессора и другую архитектуру компьютера. Здесь имеется в виду перенос с умеренными трудозатратами, не требующий полной переработки системы.

Свойство мобильности не столь однозначно положительно, как может показаться. Чтобы программа была мобильна, при ее разработке следует отказаться от глубокого использования особенностей конкретной архитектуры (таких, как количество и функциональные возможности регистров процессора, нестандартные команды и т.п.). Мобильная программа должна быть написана на языке достаточно высокого уровня (часто используется язык C), который можно реализовать на компьютерах любой архитектуры. Платой за мобильность всегда является некоторая потеря эффективности, поэтому немобильные системы распространены достаточно широко.

С другой стороны, история системного программирования усеяна останками замечательных, эффективных и удобных, но немобильных ОС, которые вымерли вместе с процессорами, для которых они предназначались. В то же время мобильная система UNIX продолжает процветать четвертый десяток лет, намного пережив те компьютеры, для которых она первоначально создавалась. Примерно 5-10% исходных текстов UNIX написаны на языка ассемблера и должны переписываться заново при переносе на новую архитектуру. Остальная часть системы написана на C и практически не требует изменений при переносе.

Некоторым компромиссом являются **многоплатформенные** ОС (например, Windows NT), изначально спроектированные для использования на нескольких аппаратных платформах, но не гарантирующие возможность переноса на новые, не предусмотренные заранее архитектуры.

1.6. Основные функции и структура ОС

Согласно многолетней традиции, при рассмотрении основ функционирования ОС принято выделять четыре основных группы функций, выполняемых системой.

- **Управление устройствами**. Имеются в виду все периферийные устройства, подключаемые к компьютеру, — клавиатура, монитор, принтеры, диски и т.п.

- **Управление данными**. Под этим старинным термином сейчас понимается работа с файлами, хотя были времена, когда обращение к данным на магнитных носителях выполнялось путем указания адреса размещения данных на устройстве, а понятия файла не существовало.

- **Управление процессами**. Эта сторона работы ОС связана с запуском и завершением работы программ, обработкой ошибок, обеспечением параллельной работы нескольких программ на одном компьютере.

- **Управление памятью**. Оперативная память компьютера — это такой ресурс, которого всегда не хватает. В этих условиях разумное планирование использования памяти является важнейшим фактором эффективной работы.

Имеется еще несколько важных обязанностей, ложащихся на ОС, которые трудно втиснуть в рамки традиционной классификации функций. К ним, прежде всего, относятся следующие.

- **Организация интерфейса с пользователем**. Формы интерфейса могут быть разнообразными, в зависимости от типа и назначения ОС: язык управления пакетами заданий, набор диалоговых команд, средства графического интерфейса.

- **Защита данных**. Как только система перестает быть достоянием одного изолированного от внешнего мира пользователя, вопросы защиты данных от несанкционированного доступа приобретают первостепенную важность. ОС, обеспечивающая работу в сети или в системе разделения времени, должна соответствовать имеющимся стандартам безопасности.

- **Ведение статистики**. В ходе работы ОС должна собираться, храниться и анализироваться разнообразная информация: о количестве времени, затраченном различными программами и пользователями, об интенсивности использования ресурсов, о попытках некорректных действий пользователей, о сбоях оборудования и т.п. Собранная информация хранится в системных журналах и в учетных записях пользователей.

Для понимания работы ОС необходимо уметь выделять основные части системы и их связи, т.е. описывать структуру системы. Для разных ОС их структурное деление может быть весьма различным. Наиболее общими видами структуризации можно считать два. С одной стороны, можно считать,

что ОС разделена на подсистемы, соответствующие перечисленным выше группам функций. Такое деление достаточно обосновано, программные модули ОС действительно в основном можно отнести к одной из этих подсистем. Другое важное структурное деление связано с понятием **ядра** системы.

Ядро, как можно понять из названия, это основная, «самая системная» часть операционной системы. Имеются разные определения ядра. Согласно одному из них, ядро — это **резидентная** часть системы, т.е. к ядру относится тот программный код, который постоянно находится в памяти в течение всей работы системы. Остальные модули ОС являются **транзитными**, т.е. подгружаются в память с диска по мере необходимости на время своей работы. К транзитным частям системы относятся:

- **утилиты** (utilities) — отдельные системные программы, решающие частные задачи, такие как форматирование и проверку диска, поиск данных в файлах, мониторинг (отслеживание) работы системы и многое другое;
- **системные библиотеки подпрограмм**, позволяющие прикладным программам использовать различные специальные возможности, поддерживаемые системой (например, библиотеки для графического вывода, для работы с мультимедиа и т.п.);
- **интерпретатор команд** — программа, выполняющая ввод команд пользователя, их анализ и вызов других модулей для выполнения команд;
- **системный загрузчик** — программа, которая при запуске ОС (например, при включении питания) обеспечивает загрузку системы с диска, ее инициализацию и старт;
- другие виды программ, в зависимости от конкретной системы.

Не менее важным является определение ядра, основанное на различении режимов работы компьютера. Все современные процессоры поддерживают, как минимум, два режима: **привилегированный** режим (он же режим ядра, kernel mode) и **непривилегированный** (режим задачи, режим пользователя, user mode). Программы, работающие в режиме ядра, имеют полный, неограниченный доступ ко всем ресурсам компьютера: его командам, адресам, портам ввода/вывода и т.п. В режиме задачи возможности программы ограничены, она, в частности, не может выполнить некоторые специальные команды. Аппаратное разграничение возможностей является абсолютно необходимым условием реализации надежной защиты данных в многопользовательской системе. Отсюда вытекает и определение ядра как части ОС, работающей в режиме ядра. Все остальные программы, как системные утилиты, так и программы пользователей, работают в режиме пользователя и должны обращаться к ядру для выполнения многих системных действий.

Следует сказать, что переходы из режима пользователя в режим ядра и обратно — это действия, требующие определенного времени, и слишком частое их выполнение может привести к заметному снижению скорости работы программ. В связи с этим определение того, какие функции должны поддерживаться ядром, а какие лучше выполнять в режиме пользователя — это непростая и важная задача, которую должны решить разработчики ОС.

Особую роль в структуре системы играют **драйверы устройств**. Эти программы, предназначенные для обслуживания конкретных периферийных устройств, несомненно, можно отнести к ядру системы: они почти всегда являются резидентными и работают в режиме ядра. Но в отличие от самого ядра, которое изменяется только при появлении новой версии ОС, набор используемых драйверов весьма мобилен и зависит от набора устройств, подключенных к данному компьютеру. В некоторых системах (например, в ранних версиях UNIX) для подключения нового драйвера требовалось перекомпилировать все ядро. В большинстве современных ОС драйверы подключаются к ядру в процессе загрузки системы, а иногда разрешается даже загрузка и выгрузка драйверов в ходе работы системы.