Ядро ОС

Когда дело касается структуры операционной системы, то самое основное – это ядро. Также как в компьютере самое основное – это процессор, в операционной системе то, что составляет операционную систему – называется ядром. Существует 2 основных направления в дизайне ядер операционных систем: это **монолитное ядро** и **микроядро**. Есть много других вариаций, есть экзоядро, есть наноядро – это вариации микроядра, есть гибридное ядро – некая комбинация монолитного и микроядра, но в целом направлений – 2, и они просто диаметрально противоположны. Первое говорит, что вся операционная система должна быть одной большой программой и всё, что максимально возможно в эту программу записать, должно быть туда записано. Та грань между операционной системой и не операционной системой довольно абстрактна, и она зависит от того, какие решения приняли дизайнеры этой операционной системы, например, драйверы каких-то периферийных устройств или графический интерфейс – для каких-то операционных систем будет являться частью операционной системы, а для других они могут являться просто какими-то пользовательскими приложениями.

В монолитном ядре мы стараемся записать как можно больше главных и полезных функций в одну большую программу, которая выполняется в этом режиме ядра, в этом режиме, когда операционная система имеет максимальный доступ к ресурсам компьютера. Это в первую очередь позволяет сделать ядро довольно быстрым, потому что это одна программа: в ней всё вместе, всё рядом – и для того, чтобы, допустим, одна часть операционной системы работала совместно с другой частью, обменивались какими-то данными – им понадобится, грубо говоря, просто меньше времени на это. Но проблема здесь такая же как бывают проблемы, когда все яйца кладут в одну корзину – если что-то пойдет не так, то всё пойдет не так, если какая-то одна часть операционной системы вызовет ошибку, то вся операционная система, всё ядро может перестать работать.

Очевидным решением этой проблемы является микроядро. Вместо того, чтобы запихивать всё, что можно в одну большую программу – мы запихнем только самое необходимое, самый минимум, необходимый для работы операционной системы, и назовем это микроядром, потому что оно будет физически намного меньше. Все остальные части будут работать на следующем уровне, они будут пользоваться операционной системой, и, с точки зрения конечного пользователя, они будут казаться частью операционной системы, но, по сути, они будут такими же программами, как наши программы, которыми мы пользуемся. Это позволит минимизировать риск. Если какая-то часть сломается, вызовет ошибку, то ядро не обязательно из-за этого перестанет работать. Например, в Linux, во многих Unix-системах графическая часть не входит в ядро[1], поэтому, если что-то пошло не так с вашими окнами и кнопками, то операционная система за-за этого не умрет, она продолжит работать и зависнувший графический интерфейс можно будет просто уничтожить и перезапустить – ядро продолжит работать. Минус, естественно, это то, что было плюсом в монолитном ядре – это скорость. Теперь у нас бóльшее расстояние между частями операционной системы, им приходится друг с другом общаться с помощью какого-то интерфейса, им приходится посылать сигналы и так далее. Это немного замедляет всё.

Примечание:
[1] Несмотря на то, что в Linux графическая часть не входит в состав ядра, микроядром оно не является.

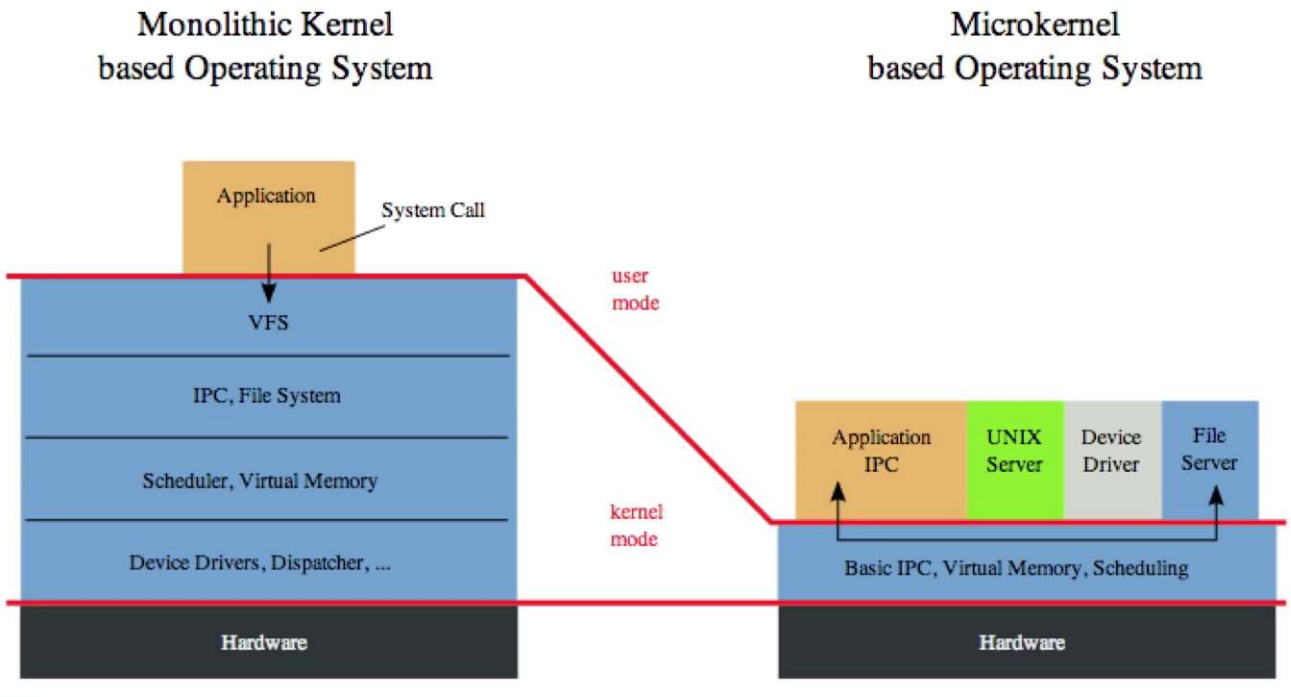

Рисунок 6. Монолитное ядро vs. микроядро

На рисунке 6 хорошо видна концепция размеров ядра – это то, что может входить в монолитное ядро и то, что входит в микроядро, и также хорошо видна концепция режимов работы: режим ядра и пользовательский режим. В монолитном ядре в режиме ядра работают намного больше программ как фундаментальной части операционной системы, которые должны быть, так и довольно высокого уровня абстракции – вещи, вроде файловой системы, виртуальной памяти и так далее. В микроядре в режиме ядра работают только базовые программы, базовый код – всё остальное работает в пользовательском режиме, что, собственно, немного улучшает безопасность.