Монолитная или макроядерная ОС представляет собой большую программу, выполняющуюся в привилегированном режиме. (схем1)

*Уровень ядра для привилегированных команд. Это отличает их от выполнения команд на уровне пользователя.*

Компонент такой ОС являются частями одной программы, а не сам. модулями.

В монолитных ОС сборка ядра осуществляется для каждого компьютера, на который устанавливается ОС. При этом можно выбрать список компонентов, которые могут быть включены в ядро. Единственный способ добавить или удалить компоненты это собрать ядро заново.

Присутствие лишних компонентов в ядре является крайне нежелательным, т.к. ОС всегда полностью загружается в оперативную память.

Т.е. все эти программы уже в памяти. Чем больше

Микроядерные ОС

В них ядро более компактное за счет того, что часть его функций делегируется различным исполняемым службам, которые выполняются в режиме пользователя. (схем2)

Примеры: службы сетевого доступа, файловых систем, драйвера устройств.

*Для взаимодействия с аппаратной часть службы должны обращаться к ядру, что замедляет работу с оборудованием, то прямого доступа у служб нет.*

В МикроЯ ОС установка компонентов не прерывает ее работы. Такие ОС более надежны, поскольку ошибка в режиме пользователя менее опасна чем ошибка в режиме ядра.

Однако в некоторых случаях использование МикроЯ ОС приводит к замедлению выполнения системных вызовов по сравнению с архитектурой монолитных ОС.

Гибридные ОС (ГОС)

В большинстве случаев современные ОС используют различные комбинации МикроЯ и Монолитной архитектуры. (схем3)

+ маленькое ядро, не требует много памяти.

+ различные функции реализованы в исполняемых службах, можно не переустанавливать ОС при добавлении новых компонентов

+ часть исполняемых служб в режиме ядра

- противоположны

Современные ОС

Все современные UNIX- подобных.

POSIX/

* Linux это не ОС, это ядро. На его основе: Debian (ноутбуки, серверы). Ubuntu. Fedora. Red Hat enterprise Linux (для корпоративных решений). Mandriva . OpenSUSU. Android. (интерфейс GNOME, KDE).
* Oracle Solyaris (от Sun Microsystem) – для корпоративных решений.
* Apple IOS, MacOS
* Windows
  + Для устройств Windows Phone
  + Для ПК
  + Для серверных приложений
* IBM AIX – для суперкомпьютеров

Процессы и потоки

Процессор переключается между программами, предоставляя каждой свое время. В каждый конкретный момент времени занят только одной программой. Но при этом создается иллюзия параллельного выполнения всех программ.

Процесс

Все многозадачные ОС используют модель процессов. Согласно этой модели выполняемые программы, организуются в виде процессов.

Процесс это абстрактное понятие, описывающее работу программы.

Для реализации используется ОС специальная таблица : ТАБЛИЦА процессов.

Каждый элемент такой таблицы содержит следующую информацию:

1. PID – идентификатор процесса.
2. ID пользователя от имени которого запущен процесс.
3. Значение сегментных регистров – сегмент данных, сегмент кода, сегмент стека – CS, DS и т.д.
4. Значение регистров данных процессора – AX аккумулятор, CX – регистр счетчика, DX – регистр данных.
5. Значение счетчика команд
6. Значение базового указателя
7. Состояние процесса
8. Приоритет процесса
9. Использованное процессорное время
10. Доп. Информация:

Восстановление с точки прерывания. По той информации, которая была сохранена в таблице (хранится в ядре).

Создание процесса

При загрузке ОС создается несколько системных процессов. Кроме того, уже созданные процессы могут создавать новые процессы. Для этого созданный процесс выполняет специальный системный вызов, который заставляет ОС создать новый процесс. В различных ОС для создания новых процессов используются различные подходы.

В ОС linux, solari, IOS используется подход предложенный в стандарте POSIX (переносимый интерфейс ОС UNIX).

Согласно этому стандарту с помощью функции fork, которая создает копию вызывающего процесса. В дальнейшем созданный процесс может продолжить выполнять ту же программу или же вызывать функцию exec для загрузки и выполнения новой программы.

В MW преимущество отдается другой функции CreateProcec=ss, которая создает вызывает новую программу. При это создается новое адресное пространство.

Завершение

Завершается в одно из:

1. Завершила работу и выполнени=ия спец. вызов, который завершает процесс: ExitProcess, exit
2. Произошла неустранимая ошибка в результате которой программ более не может исполнять свою работу.
3. Другой процесс выполнил системный вызов, в результате которого был принудительно завершен указанный процесс. Радикально: В POSIX – kill. В Win TerminateProcess. При завершении процесса удаляется созданное для него адресное пространство, однако ОС не сразу удаляет из таблицы запись о совершенном процессе Благодаря этого какое-то время все еще можно узнать информацию о операции: сколько процессорного времени.

(!) Если программа завершается совсем, но не возвращаются дескрипторы.