Титульник

Здравствуйте, меня зовут Сластухин Антон, я являюсь студентом КБ-4.

На данный момент я занимаюсь изучением проблемы создания операционной системы на базе микросервисной архитектуры.

План Доклада

1. Познакомимся с микросервисной архитектурой
2. Сравним её с монолитной
3. Рассмотрим MINIX 3 в качестве примера микросервисной ОС
4. Проведём исторический экскурс, в ходе которого прояснится текущую ситуацию на рынке ОС

1. MicroServiceArchitecture

*Микросервис* – отдельная, независимая, выполняющая свою задачу, составная единица приложения.

Обладает следующими свойствами:

* Небольшой
* Узкоспециализированный
* Независимый
* Закрытый

О каждом подробнее

Небольшой – На полное переписывание микросервиса уйдёт

* не более 1 итерации Agile(1-7дней),
* на поддержку микросервиса требуется не более 12 разработчиков, каждый их которых может быть задействован в других микросервисах.

*Узкоспециализированный -* решает одну задачу, находится в таком бизнес-контексте, что целиком помещается в голове одного человека

*Независимый* – взаимодействует с другими сервисами через заданные интерфейсы. Это позволяет вести разработку независимо от разработчиков других сервисов. Более того, это избавляется от неявных зависимостей с другими модулями.

*Закрытый* – реализация недоступна из вне. Бизнес контекст и служебная информация скрыты на уровне сервиса.

2.Сравнение микросервиса и монолита

3. MINIX

MINIX, написанная в 1987 году, [Эндрю Таненбаум](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%BD%D0%B5%D0%BD%D0%B1%D0%B0%D1%83%D0%BC,_%D0%AD%D0%BD%D0%B4%D1%80%D1%8E)ом является эталоном микросервисных ОС, поэтому рассмотрим её в качестве первоочерёдно.

Операционная система состоит из наборов служб, типов сущностей, множеств операций над ними – всё это формирует пользовательский интерфейс. Пользователями выступают программисты, которые используют возможности

1.Микроядро отвечает за низкоуровневое управление памятью, аппаратные прерывания и планирование процессов.

2.Процессы могут общаться между собой в, преимущественно, синхронном режиме, отправляя сообщения. При этом, не все процессы могут обмениваться сообщениями между собой.

Зачастую системный вызов состоит из отправки и получения сообщения. Кажется, что это негативно сказывается на эффективности системы, но нет, на поддержания подобного взаимодействия уходит около 1% CPU, что несущественно. Более того, подобная структура (когда ядро не участвует в взаимодействии процессов) позволяет исключить проблемы переполнения буффера ядра и частично решает проблему взаимоблокировок процессов.

3.Драйверы тоже представляются изолированными процессами, за тем исключением, что им разрешено обращаться к сервисам ядра, например, чтобы записать значения в аппаратные порты ввода/вывода или осуществить копирование информации из одного процесса в другой. При этом ядро точно знает, какие порты может использовать драйвер и какие процессы ему доступны.

Пользовательский интерфейс ОС формируется набором серверов (сервисов). Выделим основные: Файловый сервер, диспетчер процессов, сервер реинкарнации, подсистема логирования, восстановления, клонирования и т.д.

Пользовательские процессы(серверы) взаимодействуют c основными серверами как с API.

4.Ограничения

По ходу развития программирования постепенно накапливались программные ограничения, которые ограничивали деятельность пользователя/программиста для увеличения устойчивости/безопасности системы.

Пример:

1.С появлением диспетчеров памяти (MMU) пользовательские приложения потеряли возможность напрямую обращаться к устройствам ввода-вывода, теперь это нужно делать только системные вызовы.

Многие были против подобных ограничений, ссылаясь на понижение эффективности, но в последствии пришли к соглашению, что подобные меры решают проблему аварийной остановки системы в случае ошибки в пользовательских программах.

2.Дейкстра о GOTO

5.Ограничения в MINIX 3

Аналогично MINIX 3 ограничивает разработчиков больше, чем другие ОС.

1. Изоляция ядра – каждый сервис имеет свою битовую карту того, что какие системные вызовы он может использовать.
2. Изоляция памяти – каждый сервис работает в изолированном от других пространстве. В случае обнаружения попыток незаконно обратиться к области памяти другого сервиса сервер реинкарнации уничтожает и переподнимает нарушителя. Законно обратиться к области памяти другого процесса можно, если он дал явное разрешение или этот процесс обладает таким правом, выданным системой (является надёжным процессом).

Ограниченное межпроцессорное взаимодействие – каждый процесс имеет список разрешённых сервисов, с которыми он может общаться, за контроль

6.Преимущества MINIX 3

Во многом преимущества модели программирования в MINIX обусловлены преимуществами микросервисной архитектуры.

Перечислим основные преимущества:

1. Короткий цикл разработки – за счёт небольшой команды и небольшого, в сравнении с кодом ОС, кода сервиса.
2. Высокая производительность – ошибка в драйвере (или обновление) не вынуждает выполнять перезагрузку системы, логи позволяют понять, что произошло, а сервер реинкарнации перезапускает процесс.
3. Разработка упрощена в сравнении с монолитными системами за счёт того, что разработчик ограничен методами интерфейсов, он не задумывается над внутренней реализацией.
4. Высокая отслеживаемость – за счёт того, что каждый сервис изолирован и работает в среде пользователя, ошибка возникает не ядре, а в среде пользователя – её можно проанализировать средствами отладки, более того, сервисы логируют свои действия дополнительно на лог сервер. За все ошибки работы сервиса отвечает его разработчик, производитель.
5. Гибкость конфигурации – за счёт того, каждый элемент системы мал и изолирован, можно сконфигурировать систему под конкретные задачи. Например: можно подменить сервис, отвечающий за файловую систему, за подсистему логирования, при этом работа остальных сервисов не нарушится.
6. Облегчение поддержки – мечта системного администратора, все модули малы, независимы и понятны.

7.Выводы по MINIX 3

MINIX 3 – модульная операционная система, построенная быть сверхнадёжной, это свойство обеспечивается более жёсткими ограничениями, чем у других ОС. Система гибкая, что позволяет сконфигурировать её под различные задачи, при этом, обладает широкими возможностями администрирования, что делает её обслуживание удобным. Модель программирование мало отличается от привычной модели программирования, но имеет ряд преимуществ, обусловленных особенностями микросервисной архитектуры – простотой, изолированность, эффективностью, расширяемостью. Программист для себя найдёт удобную почву для разработки, которая будет подталкивать его к написанию более строго, соответственно, более эффективного кода. Пользователь получит безопасную, конфигурируемую, высокоэффективную систему.

Я считаю, что микросервисные ОС, такие, как MINIX, должны были вытеснить монолитные ОС с рынка, но этого не произошло. Попробуем разобраться в причинах.

7.Сравнение путей развития MS-DOS, Linux и MINIX

1. MS-DOS (1981г) была разработана для компьютера IBM PC компании IBM, которая в 1975 году выпустила первый массовый персональный компьютер (ПК) – IBM 5100, но он оказался слишком дорог для обычных пользователей. Целью компании в том время было занять часть перспективного развивающегося рынка – поэтому компания выпустила компьютер IBM PC, с операционной системой MS-DOS, которая была доработана компанией Microsoft специально под требование IBM из ОС 86-DOS, права на которую были выкуплены.

Как мы видим, MS-DOS изначально ориентирована под конкретный компьютер с конкретным процессором (Intel 8088) – это и есть «низкая переносимость».

1. Linux начал разрабатывать Линус Торвальдс для своего [персонального компьютера](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) с [процессором](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) [Intel 80386](https://ru.wikipedia.org/wiki/Intel_80386), который на тот момент имел очень популярную архитектуру. В последствии Linux развивался как продукт с открытым исходным кодом, открытым сообществом разработчиков. Разные мнения в последствии привели к появлению разных дистрибутивов на базе Linux.
2. Minix же, напротив, развивался не как рыночный продукт, не как открытый продукт, развиваемый сообществом – это была, в первую очередь, учебная операционная система.

Принципиально то, что MS-DOS, развиваемая большой компанией как закрытый продукт под конкретную архитектуру с целью выхода на новый широкий рынок персональных компьютеров обрела высокую популярность среди пользователей из-за своей доступности и функционала, а Linux пошёл по другому пути – по пути развития открытого продукта широким сообществом и тоже обрёл популярность, но Minix, скорее всего, опоздал начать борьбу за рынок слишком поздно.