Колледж Автономной некоммерческой образовательной организации высшего образования

«Научно-технологический университет «Сириус»

Учебная дисциплина

«Введение в специальность»

Реферат

Принципы работы и история создания ядра Linux

Работу подготовил:

ФИО, группа

Студент группы К0709-24/1

Далингер Иван Алексеевич

Проверила:

Яковлева Софья Вячеславовна

Оглавление

[Оглавление 2](#_Toc192456798)

[Введение 3](#_Toc192456799)

[1 История развития ядра Linux 4](#_Toc192456800)

[2 Архитектура ядра Linux 6](#_Toc192456801)

[2.1 Ядро 6](#_Toc192456802)

[2.2 Модули ядра 6](#_Toc192456803)

[2.3 Начальный образ загрузки 6](#_Toc192456804)

[2.4 Init 7](#_Toc192456805)

[2.5 Командная оболочка 7](#_Toc192456806)

[2.6 Безопасность 7](#_Toc192456807)

[3 Принцип загрузки систем на ядре Linux 9](#_Toc192456808)

[Вывод 11](#_Toc192456809)

[Список литературы 12](#_Toc192456810)

# Введение

Ядро является центральным компонентом любой операционной системы. Существуют различные точки зрения на то, что именно составляет операционную систему. Некоторые исследователи приравнивают операционную систему к ее ядру, в то время как другие включают в это понятие как ядро, так и системные программы, обеспечивающие взаимодействие пользователя с аппаратными ресурсами через ядро.

Ядро операционной системы представляет собой низкоуровневую программу, которая абстрагирует сложность аппаратного обеспечения и предоставляет удобный интерфейс для программ и пользователей. Аппаратное обеспечение оперирует сигналами, регистрами и секторами, в то время как программы используют команды высокого уровня, такие как "записать", "прочитать", "сложить" и "вычесть". Ядро выступает в роли посредника, переводя команды программ на язык аппаратного обеспечения.

Ядро операционной системы является фундаментальным компонентом, обеспечивающим эффективную работу программного обеспечения на аппаратном обеспечении. Оно выполняет ряд критически важных функций, включая абстракцию аппаратного обеспечения, управление ресурсами, процессами, памятью и вводом-выводом. Понимание роли и функций ядра необходимо для понимания принципов работы операционных систем.

# 1 История развития ядра Linux

Начало проекту было положено в 1991 году с публикации сообщения в новостной группе.

К тому времени GNU уже создал множество составляющих для свободной операционной системы, но её ядро GNU Hurd ещё не было готово. Поэтому пустующее место ядра для свободной операционной системы занял Linux и, несмотря на ограниченную функциональность ранних версий, привлёк к себе множество разработчиков и пользователей.

«Linux» как ядро операционной системы, разработка которого была начата Торвальдсом — лишь небольшая часть многих из использующих его систем, которые обычно тоже называют «Linux». Это иногда приводит к путанице, в связи с чем существует спор об именовании GNU/Linux — некоторые сторонники движения GNU считают, что именно такое наименование является корректным для операционной системы на базе ядра Linux и использующей наработки проекта GNU (такие, как glibc, gcc, bash, coreutils и другие).

Хронология:

* Апрель 1991: 21-летний Линус Торвальдс начал работу над некоторыми механизмами операционной системы. Он начал с эмулятора терминала и планировщика задач.
* 25 августа 1991: Торвальдс поместил сообщение в новостную группу по MINIX.

«*От: torvalds@klaava.Helsinki.Fi (Линус Бенедикт Торвальдс)  
Новостная группа: comp.os.minix  
Тема: Небольшой опрос о моей новой операционной системе  
Message-ID:<1991Aug25.205708.9541@klaava.Helsinki.Fi>  
Дата: 25 Aug 91 20:57:08 GMT  
Организация: Хельсинкский Университет*

*Привет всем тем, кто использует миникс —*

*Я делаю (свободную) операционную систему (это только хобби, не столь большое и профессиональное, как GNU) для 386 (486) AT-клонов. Эта система пишется с апреля и скоро будет готова. Я хочу получить любой отзыв, касающийся вещей, которые нравятся/не нравятся людям в миниксе, так как моя ОС похожа на неё (такое же устройство файловой системы (по практическим соображениям), среди прочего).*

*В настоящее время я портировал bash (1.08) и gcc (1.40), и, похоже, эти программы работают. Это значит, что я получу что-то практичное в ближайшие несколько месяцев, и я хочу узнать, какие возможности хочет большинство людей. Любые предложения принимаются, но я не обещаю, что я осуществлю их :-)*

*Линус (torvalds@kruuna.helsinki.fi)*

*P. S. Да — в ней нет кода миникс, и будет мультипотоковая ФС. Система НЕПЕРЕНОСИМА (использует команды Intel 386 и т. д.) и, вероятно, будет поддерживать только жёсткие диски AT, так как это всё, что у меня есть :-(»*

* 17 сентября 1991: Linux версии 0.01 (10 239 строк кода).
* Декабрь 1991: Linux версии 0.11. Это была первая версия Linux, на которой можно было собрать Linux из исходных кодов.
* 19 января 1992: Первое сообщение в группе новостей alt.os.linux.
* 31 марта 1992: Создана специализированная группа новостей comp.os.linux.
* Апрель 1992: Linux версии 0.96, на котором стало возможно запустить графический сервер X Window System.
* Весь 1993 и начало 1994: 15 тестовых выпусков версии 0.99.\* (в июле 1993 введено понятие BogoMips).
* 14 марта 1994: Linux версии 1.0.0 (176 250 строк кода).
* 9 мая 1996: Выбран символ Linux — пингвин Tux.
* 9 июня 1996: Linux версии 2.0.0 (777 956 строк кода).
* 25 января 1999: Linux версии 2.2.0, изначально довольно недоработанный (1 800 847 строк кода).
* 4 января 2001: Linux версии 2.4.0 (3 377 902 строки кода).
* 18 декабря 2003: Linux версии 2.6.0 (5 929 913 строк кода).
* 23 марта 2009: Linux версии 2.6.29, временный символ Linux — тасманский дьявол Tuz (11 010 647 строк кода).
* 22 июля 2011: релиз Linux 3.0 (14,6 млн строк кода).
* 23 февраля 2015: первый релиз-кандидат Linux 4.0 (более 19 млн строк кода).
* 7 января 2019: первый релиз-кандидат Linux 5.0 (более 26 млн строк кода).

# 2 Архитектура ядра Linux

У Linux архитектура модульная – то есть какой-то функционал вынесен в модули и подгружается по необходимости.

## 2.1 Ядро

Также Linux называют монолитным – потому что всё что делает ядро происходит в рамках одной программы – а правильнее сказать – все части ядра работают в одном адресном пространстве. Помните, мы обсуждали, что это такое, когда говорили о процессах? Но так как у Linux-а огромный функционал, который бессмысленно держать одновременно в памяти – поэтому функционал вынесен в модули, благодаря чему ускоряется работа ядра.

Ядро операционной системы, созданное, чтобы работать с POSIX-совместимыми окружениями. Обычно лежит в /boot/ и содержит в названии слово vmlinuz, где «vm» напоминает нам о поддержке виртуальной памяти, а «z» указывает, что файл сжат.

## 2.2 Модули ядра

Ядро Linux, ядро является монолитным. Это значит, что весь исполняемый код сосредоточен в одном файле. Такая архитектура имеет некоторые недостатки, например, невозможность установки новых драйверов без пера сборки ядра. Но разработчики нашли решение и этой проблеме, добавив систему модулей.

Ядро Linux позволяет драйверам оборудования, файловых систем, и некоторым другим компонентам быть скомпилированными отдельно - как модули, а не как часть самого ядра. Таким образом, вы можете обновлять драйвера непере собирая ядро, а также динамически расширять его функциональность. А еще это значит, что вы можете включить в ядре только самое необходимое, а все остальное подключать с помощью модулей. Это очень просто.

Модули ядра Linux собираются только под определенную версию ядра, есть способ запуска модуля независимо от версии ядра, если они совместимы с помощью dkms, но об этом мы поговорим позже.

Находятся все модули в папке /lib/modules/. Учитывая, что модули рассчитаны только для определенной версии ядра, то в этой папке создается отдельная подпапка, для каждой установленной в системе версии ядра. В этой папке находятся сами модули и дополнительные конфигурационные файлы, модули отсортированы по категориям.

## 2.3 Начальный образ загрузки

Начальный образ загрузки известен так же, как initrd и initramfs. Представляет собой архив с образом файловой системы, развёртываемой в оперативную память в начале процесса загрузки. Несёт в себе различные драйверы и скрипты, позволяющие инициализировать оборудование и смонтировать файловые системы.

Содержимое начального образа загрузки зависит от версии ядра и потребностей пользователя (кто-то использует ZFS, а у кого-то корень зашифрован LUKS). Поэтому образ не поставляется в дистрибутивах. В дистрибутивах поставляются фреймворки для создания начальных образов по мере необходимости. Так, обычно создание свежего образа инициируется при обновлении ядра.

## 2.4 Init

Система инициализации является ключевым компонентом операционной системы, ответственным за запуск и управление процессами. Она запускается первой при загрузке системы и имеет идентификатор процесса (PID) 1. Система инициализации определяет уровень запуска системы и управляет жизненным циклом большинства служб.

## 2.5 Командная оболочка

Командная оболочка (shell) — это интерфейс командной строки, необходимый для взаимодействия с операционной системой. Она играет ключевую роль в обеспечении работоспособности системы, так как стандартизирована в POSIX.

При входе пользователя в систему запускается «оболочка входа» (login shell), которая инициализирует настройки и переменные окружения из конфигурационных файлов. Все последующие процессы запускаются в контексте этой оболочки.

Наиболее распространённые командные оболочки:

* Bourne shell (sh):
  + Стандартная оболочка, присутствующая практически во всех дистрибутивах.
* Bourne again shell (bash):
  + Оболочка по умолчанию в большинстве дистрибутивов GNU/Linux.
  + Предоставляет расширенные возможности по сравнению с sh.
* Debian Almquist shell (dash):
  + Компактная и быстрая оболочка, совместимая с sh.
  + Используется в Debian, где /usr/bin/sh указывает на неё.
* Z shell (zsh):
  + Оболочка, схожая с bash, но с дополнительными функциями для интерактивного ввода.
  + Часто доступна в репозиториях, но не устанавливается по умолчанию.
* BusyBox:
  + Утилита для встраиваемых систем, включающая POSIX-совместимую оболочку (вызывается командой busybox sh).

## 2.6 Безопасность

PAM — Pluggable Authentication Modules — модульная система авторизации. Отвечает, как понятно из названия, за авторизацию пользователей в системе, причём разными способами. Через PAM авторизуются в том числе доменные пользователи, в таком случае PAM действует в связке с имплементацией Kerberos (обычно MIT'овский krb5), поскольку сам по себе PAM не работает с удалёнными клиентами. Модули представляют собой разделяемые библиотеки (исполняемые файлы с суффиксом so) и позволяют делать интересные штуки при входе пользователя. Например, можно создавать домашнюю директорию при первом входе (pam\_mkhomedir.so) или монтировать файловые системы (pam\_mount.so).

Классическая утилита su и более молодая sudo предназначены для исполнения команд от имени другого пользователя (по умолчанию root). Наиболее значимая разница — su требует пароль пользователя, из-под которого вы хотите работать, а sudo — ваш пароль, позволяя работать от себя, сменив контекст безопасности на другого пользователя без login. sudo гибко настраивается, позволяя запускать только определённые команды определённым пользователям из-под других определённых пользователей, как-то так.

Менеджер авторизации Polkit позволяет непривилегированным процессам взаимодействовать с привилегированными. По сути, он похож на sudo, но обладает превосходящей гибкостью и предназначен в первую очередь для приложений, в то время как sudo — утилита для пользователя. Правила пишутся, внезапно, на JavaScript'е.

Linux Security Modules (LSM) — фреймворк внутри ядра Linux, позволяющий накладывать на систему дополнительные модели безопасности. Это достигается при помощи модулей безопасности, не путать с модулями ядра. Наиболее популярные модули безопасности — SELinux и AppArmor.

# 3 Принцип загрузки систем на ядре Linux

1. **BIOS**. Отвечает за базовый ввод/вывод данных с устройств/на устройства. Делает некоторые проверки целостности устройств. К тому же, за тестирование работоспособности электроники отвечает POST (Power-on self-test, он же «тест на адекватность себя самого», выполняющийся как этап пре-загрузки), который управляется BIOS. Ищет, загружает и выполняет программу-загрузчик ОС. Берет загрузчик из твердотельного или жесткого диска. Во время загрузки BIOS'а вы можете нажать на кнопку (обычно это F12 или F2 или Del, зависит от платформы), если вам требуется внести некоторые изменения касательно настройки железа. Как только загрузчик был обнаружен и загружен в память, BIOS передает управление ему. Короче говоря, BIOS загружает и выполняет загрузочную запись (MBR).
2. **MBR**. Главная загрузочная запись, хранящаяся на жестком диске. Она размещена в 1-м секторе загрузочного диска, например /dev/hda или /dev/sda. MBR занимает меньше, чем 512 байтов. Она состоит из трех компонентов: 1) главная загрузочная информация, «живущая» в первых 446 байтах; 2) информация о таблице разделов — в следующих 64 байтах; 3) и последние 2 байта нужны для проверки корректности mbr. Она содержит информацию о GRUB'е (или LILO). Простыми словами — MBR загружает и выполняет загрузчик GRUB. Так же появились более новые технологии загрузки GPT (GUID Partition Table) — стандарт формата размещения таблиц разделов на физическом жестком диске. Он является частью Extensible Firmware Interface — стандарта, предложенного Intel на смену BIOS. EFI использует GPT там, где BIOS использует Главную загрузочную запись.
3. **GRUB** — Grand Unified Bootloader. Если в вашей системе установлено более, чем одно ядро, у вас есть возможность выбирать, которое из них должен выполняться. GRUB отображает красивую анимацию plymouth заставку, и, подождав несколько секунд интерактивного воздействия пользователя, если он не нажал ни одной клавиши, он загружает ядро, установленное по умолчанию в файле конфигурации grub. GRUB понимает, что такое файловая система (древние загрузчики Linux'а, например, LILO этого не понимают). Конфигурационный файл Grub обычно лежит по пути /boot/grub/grub.conf (так же /etc/grub.conf может быть символьной ссылкой на него). В качестве примечания к информации выше, конфигурационный файл содержит путь к ядру и образу initrd. Если быть кратким, GRUB просто напросто загружает и выполняет образы ядра и initrd.
4. **Ядро или Kernel**. Монтирует файловую систему в соответствии с настройкой «root=» в фале grub.conf. Выполняет программу /sbin/init. Поскольку init — это первый процесс, запущенный ядром Linux, поэтому она имеет идентификатор процесса (PID) №1. Можете выполнить «ps -ef | grep init» и убедиться в этом. initrd — это Initial RAM Disk, он же временный диск в оперативной памяти. initrd используется самим ядром в качестве временной корневой файловой системы, пока kernel не загрузится в реальную примонтированную файловую систему. Этот временный диск также содержит необходимые для загрузки драйверы, позволяющие получить доступ к разделам дисков и другому оборудованию.
5. **Init**. Смотрит в файл /etc/inittab для того, чтобы определить уровень выполнения (run level). Есть 6 уровней выполнения от 0 до 6. Init определяет уровень выполнения по умолчанию исходя из /etc/inittab и использует его для загрузки всех необходимых программ. Выполните «grep initdefault /etc/inittab» на вашей системе, и вы узнаете, какой уровень по умолчанию у вас используется. Если у вас не получается жить спокойно, то можете установить стандартный уровень на 0 или 6. :). В большинстве случаев вам будет достаточно уровня 3 или 5.
6. **Уровень выполнения программ (Runlevel).** Когда Линукс выполняет свою загрузку, вы можете наблюдать загрузку различных служб. К примеру, это могут быть сообщения типа «starting Postfix … OK» (запускается Postfix). Эти службы — и называются программами уровня выполнения, выполняемые из директории, которая соответствует нужному уровню выполнения. Исходя из настроек по умолчанию, система будет выполнять файлы в соответствии с нижеприведенными директориями.
   1. Выполнение уровня 0 – /etc/rc.d/rc0.d/
   2. Выполнение уровня 1 – /etc/rc.d/rc1.d/
   3. Выполнение уровня 2 – /etc/rc.d/rc2.d/
   4. Выполнение уровня 3 – /etc/rc.d/rc3.d/
   5. Выполнение уровня 4 – /etc/rc.d/rc4.d/
   6. Выполнение уровня 5 – /etc/rc.d/rc5.d/
   7. Выполнение уровня 6 – /etc/rc.d/rc6.d/

# Вывод

В заключение, ядро Linux представляет собой выдающийся пример успешного открытого программного обеспечения. Его открытость и доступность для модификации позволяют практически любому разработчику внести свой вклад в его развитие.

Кроссплатформенность ядра Linux — его способность работать на множестве различных аппаратных архитектур, таких как x86, x86-64, PowerPC, ARM, RISC-V, SPARC, IBM System/390, MIPS и других — делает его чрезвычайно гибким и универсальным. Эта особенность позволяет использовать Linux в самых разнообразных устройствах, от настольных компьютеров и ноутбуков до мобильных телефонов, планшетов, носимой электроники, сетевого оборудования, серверов и устройств интернета вещей (IoT).

# Список литературы

1. 6 шагов загрузки Linux на пальцах -   
   <https://habr.com/ru/articles/113350/>
2. Kernel (Русский) - <https://wiki.archlinux.org/title/Kernel_(%D0%A0%D1%83%D1%81%D1%81%D0%BA%D0%B8%D0%B9)>
3. Анатомия GNU/Linux –
4. <https://habr.com/ru/articles/531872/>
5. Модули ядра Linux –
6. <https://losst.pro/moduli-yadra-linux>
7. Ядро Linux -<https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_Linux#%D0%98%D1%81%D1%82%D0%BE%D1%80%D0%B8%D1%8F>
8. Ядро Linux -<https://basis.gnulinux.pro/ru/latest/basis/21/21._%D0%AF%D0%B4%D1%80%D0%BE_Linux.html>