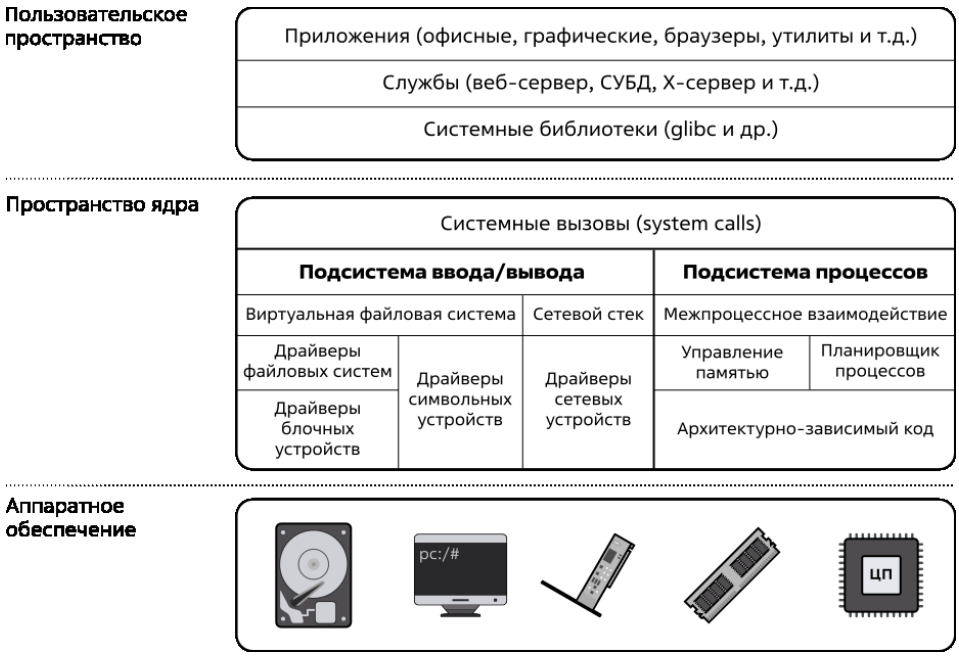
1. Архитектура ОС GNU/Linux

Задача ОС — обеспечить другому программному обеспечению аппаратно независимый интерфейс к устройствам, а также предоставить инструменты и пользовательский интерфейс для проведения работ по настройке и сопровождению ОС.

Архитектура ОС состоит из следующих компонентов:

- пользовательское пространство — пространство, где приложения пользователя могут получить доступ к ресурсам компьютера через системные вызовы ядра;
- пространство ядра — состояние ядра системы с повышенными правами и защищённое пространство памяти с полным доступом к оборудованию устройства;
- аппаратное обеспечение — интеграция ядра ОС с аппаратным обеспечением, которое позволяет эффективно использовать доступные ресурсы и обеспечивает стабильную работу на самых разных устройствах.



Прикладные (пользовательские) процессы работают в непривилегированном режиме процессора. Они используют виртуальные адреса памяти, которые для активных процессов отображаются в физические адреса с помощью устройства управления памятью (Memory Management Unit, MMU) ядром ОС.

Процессы ядра функционируют, когда центральный процессор (ЦПУ, CPU) компьютера находится в привилегированном режиме. Только процессы, работающие в привилегированном режиме процессора, имеют непосредственный доступ к аппаратным ресурсам и имеют доступ к физическим адресам оперативной памяти.

Ядро Linux относится к классу монолитных ядер, которые содержат весь функционал, необходимый для обеспечения взаимодействия прикладных процессов с аппаратными устройствами:

- ☐. Системные вызовы (System Calls): Ядро предоставляет набор функций, которые используются приложениями для взаимодействия с ядром. Эти функции называются системными вызовами.
- ☐. Подсистема ввода-вывода ядра Linux — это часть ядра, которая обрабатывает все операции ввода и вывода (I/O). Она включает в себя буферизацию, кэширование, драйверы устройств и файловые системы.
  - ☐. Буферизация и кэширование: Ядро Linux включает буферы и кэши для оптимизации операций ввода-вывода. Это увеличивает производительность, так как обращения к диску обычно намного медленнее, чем обращения к оперативной памяти.
  - ☐. Драйверы устройств: Чтобы взаимодействовать с аппаратными устройствами, такими как жесткие диски, принтеры, сетевые карты и т.д., ядро Linux использует программы, называемые драйверами устройств. Этот код составляет значительную часть ядра и служит для обмена данными между ядром и устройствами.
  - ☐. Файловые системы: Подсистема ввода-вывода ядра Linux также отвечает за работу с файловыми системами. Файловая система — это способ организации данных на устройствах хранения. Linux поддерживает множество типов файловых систем, включая, но не ограничиваясь: ext2, ext3, ext4, XFS, Btrfs, NFS, CIFS, и многие другие.
  - ☐. Блоки и символы устройств: Устройства в системе классифицируются как блочные или символьные. Блочные устройства (например, жесткие диски) обрабатывают данные в блоках, в то время как символьные устройства (например, клавиатура) работают с данными по одному символу за раз.
  - ☐. Сетевой стек (Network Stack): Это набор программного обеспечения в ядре, который обрабатывает сетевые операции.
- ☐. Подсистема процессов ядра в Linux отвечает за управление процессами, выделение процессорного времени, синхронизацию процессов, управление памятью и многое другое.
  - ☐. Межпроцессное взаимодействие (IPC): Процессы часто должны взаимодействовать друг с другом или работать совместно над задачей. Linux предоставляет множество механизмов синхронизации, таких как семафоры, мьютексы и барьеры.
  - ☐. Планировщик процессов: Это часть подсистемы процессов

ядра, которая определяет, какие процессы должны быть запущены и когда. Он делит время процессора между всеми запущенными процессами.

☐. Управление памятью: Подсистема процессов ядра также отвечает за управление физической и виртуальной памятью. Она следит за тем, чтобы каждый процесс имел достаточно памяти, не превышал своего предела и не мешал другим процессам.

☐. Архитектурно-зависимый код ядра Linux — это часть ядра, которая специфична для конкретной архитектуры аппаратного обеспечения (например, x86, ARM, MIPS, PowerPC и т.д.). Хотя большая часть кода ядра Linux написана на языке C и является архитектурно-независимой, некоторые части кода нужно специально адаптировать для работы с конкретной архитектурой процессора для обеспечения соответствующего взаимодействия с аппаратным обеспечением. Это включает в себя такие вещи, как: конкретные для процессора инструкции, управление памятью, исключения, прерывания и другие низкоуровневые функции.

ПРИМЕЧАНИЕ

В отличие от монолитных ядер, микроядра содержат лишь минимально необходимую функциональность, а остальная функциональность помещается в отдельные независимые процессы. Эти процессы работают вне ядра в непривилегированном режиме выполнения процессора. Преимущество использования монолитных ядер заключается в эффективности работы, поскольку требуется меньше затратных переключений в привилегированный режим процессора и обратно. Недостатки монолитных ядер заключаются в следующем:

- исходный код монолитных ядер более сложный и запутанный, чем в микроядрах;
- для включения нового модуля требуется перекомпиляция всего исходного кода ядра, а разработка новых модулей затруднена из-за отсутствия интерфейсов у монолитных ядер;
- монолитное ядро занимает больше места в оперативной памяти.

2. Системные вызовы и системныебиблиотеки

Системные вызовы (System Calls) — это интерфейсы между пользовательским пространством и ядром Linux. Они позволяют программам в пользовательском пространстве запрашивать услуги из ядра системы. Некоторые примеры системных вызовов в Linux включают open(), read(), write(), close(), wait(), exec(), fork(), exit() и так далее.

Системные библиотеки (System Libraries) — это special-программы, которые обеспечивают пользовательские программы интерфейсами к функциям ядра. В этих библиотеках обычно содержатся множество вызовов функций, и программы могут использовать эти вызовы функций, чтобы взаимодействовать с операционной системой. Например, одной из наиболее известных системных библиотек в Linux является glibc (GNU C Library), которая обеспечивает основные интерфейсы для таких операций, как ввод/вывод файлов, обработка строк, доступ к математическим функциям и так далее.

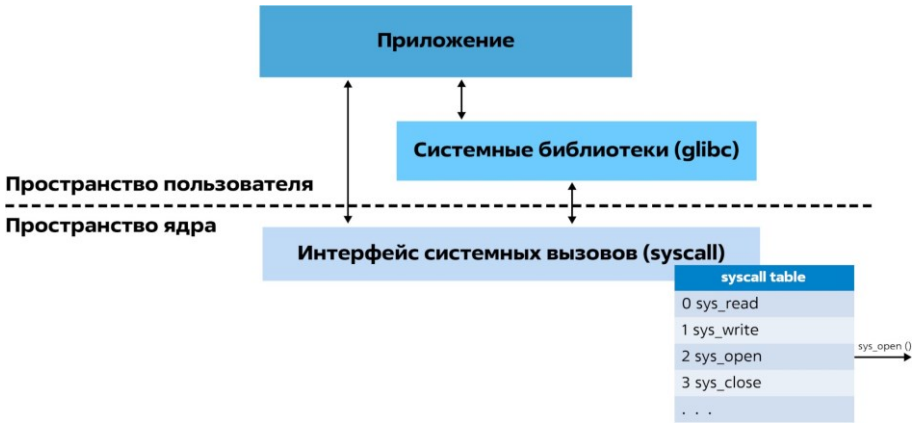
Системные библиотеки и системные вызовы взаимодействуют тесно, чтобы обеспечить надежный и эффективный интерфейс для работы приложений с операционной системой. Например:

☐. Программа (написанная, скажем, на C или C++) вызывает функцию, определенную в системной библиотеке. Например, printf() – функция стандартной библиотеки C для вывода текста на экран.

☐. Функция в системной библиотеке, в свою очередь, выполняет системный вызов, чтобы взаимодействовать с ядром Linux. В случае с printf(), библиотека стандартного ввода-вывода C может использовать системный вызов write().

☐. Ядро Linux обрабатывает системный вызов и выполняет необходимую работу. Это может включать в себя обмен данными с оборудованием, манипулирование файловой системой, контроль процессами и так далее.

☐. Если это необходимо, ядро Linux вернет результат системного вызова назад в системную библиотеку, которая затем будет передавать результаты пользовательской программе.



Таким образом, системные библиотеки служат своего рода «посредниками» между вашими приложениями и ядром ОС через системные вызовы.

Обращение к системному вызову (system call) переключает режим работы процессора из пользовательского режима в режим ядра. Программы в Linux могут вызывать System Calls прямо, минуя System Libraries, через использование механизма прерываний. Это обычно делается с использованием ассемблера.

Пример программы написан на языке ассемблера, который использует прямые системные вызовы (System Calls) для вывода сообщения "Hello, World!":

```
section .data
   ; Data to be printed
   hello db 'Hello, World!',0
   ; length of our dear string
   helloLen equ $-hello
section .text
   global _start
_start:
   ; syscall number (sys_write)
   mov eax, 4
   ; first argument: file handle (stdout)
   mov ebx, 1
   ; second argument: pointer to message to write
   mov ecx, hello
   ; third argument: message length
   mov edx, helloLen
   ; Call the kernel
   int 0x80
   ; syscall number (sys_exit)
   mov eax, 1
   ; Exit code (0)
   xor ebx, ebx
   ; Call the kernel
   int 0x80
```

Сначала программа использует системный вызов write (4), передавая файловый дескриптор stdout (1), указатель на строку и ее длину. Затем она осуществляет системный вызов exit (1) с кодом выхода 0. int 0x80 это механизм прерываний, с помощью которого происходит передача управления ядру ОС для выполнения системных вызовов. При этом eax используется для указания номера системного вызова, а другие регистры (ebx, ecx и edx) – для передачи аргументов системного вызова.

3. Комплекс средств защиты Astra Linux

В ОС Astra Linux реализована эшелонированная защита информации, которая обеспечивается с помощью:

- замкнутой программной среды — предоставляет возможность внедрения цифровой подписи в исполняемые файлы формата ELF, входящие в состав устанавливаемого СПО, и в

расширенные атрибуты файловой системы;

- мандатного контроля целостности — субъектам и сущностям задаются уровни целостности, что затрудняет программным закладкам внедрение в защищаемую ОС и дальнейшее функционирование в ней;
- мандатного управления доступа — каждому объекту доступа присваивается уровень конфиденциальности, каждому субъекту доступа присваивается уровень доступа, обеспечивающие защиту информации от несанкционированного доступа;
- дискреционного управления доступа — обеспечивает наличие для каждой пары (субъект-объект) явное и недвусмысленное перечисление разрешенных типов доступа.

Комплекс средств защиты (подсистема безопасности PARSEC) предназначен для реализации функций ОС по защите информации от НСД и предоставления администратору безопасности информации средств управления функционированием комплекса средств защиты (КСЗ).

В состав КСЗ ОС Astra Linux входят:

- Модули подсистемы безопасности PARSEC, входящие в состав ядра ОС;
- Библиотеки и модули аутентификации;
- Утилиты безопасности;
- Подсистема протоколирования (регистрации);
- Графическая подсистема и консольный вход в систему;
- Средства контроля целостности;
- Средства восстановления;
- Средства разграничения доступа к виртуальным машинам;
- Средства разграничения доступа к подключаемым устройствам.

> **ПРИМЕЧАНИЕ**
> С дополнительной информацией о комплексе средств защиты Astra Linux можно ознакомиться в документе «ОПЕРАЦИОННАЯ СИСТЕМА СПЕЦИАЛЬНОГО НАЗНАЧЕНИЯ "ASTRA LINUX SPECIAL EDITION" Руководство по КСЗ. Часть 1».

4. Уровни защищенности ОС Astra Linux

Впервые в российской практике в одном релизе Astra Linux Special Edition реализованы три уровня защиты, включая безопасную среду виртуализации — на выбор заказчика и с учётом данных о модели нарушителей и актуальных для организации угроз.

Вариант несертифицированной операционной системы Astra Linux Special Edition «Орёл» 1.7 рекомендован для применения в открытых сегментах инфраструктур, подключённых к сетям общего доступа, в образовательных учреждениях и других ИТ-системах, не требующих специальных средств защиты, а также для домашнего использования. Не может применяться в системах, обрабатывающих информацию ограниченного доступа, к которым предъявляются требования по защите информации.

Усиленный уровень защищённости («Воронеж») рекомендуется для обработки конфиденциальной информации в ГИС, в информационных системах персональных данных, а также в составе значимых объектов КИИ любого класса (уровня, категории) защищённости. Дополнительно используется в других информационных (автоматизированных) системах для обработки информации ограниченного доступа без содержания сведений, составляющих гостайну.

Максимальный уровень защищенности («Смоленск») рекомендуется для обработки информации любой категории доступа в ГИС~~, в информационных системах персональных~~ данных, в составе значимых объектов КИИ, иных информационных (автоматизированных) системах, обрабатывающих информацию ограниченного доступа, в т.ч. содержащую сведения, составляющие гостайну до степени «особой важности» включительно.

ПРИМЕЧАНИЕ

☐. Подробнее о функциях безопасности на разных уровнях защищённости можно ознакомиться в приложении Б.

☐. С дополнительной информацией можно ознакомиться по адресу https://astralinux.ru/products/astra-linux-special-edition/.

☐. Возможности реализации мер защиты информации в соответствии с приказами ФСТЭК России средствами Astra Linux Special Edition x.7: https://wiki.astralinux.ru/pages/viewpage.action?pageId=181666117.

Приложение Б. Функции безопасности Astra Linux SE на разных уровнях защищённости ОС

На каждом уровне защищённости доступны функции безопасности предыдущего уровня.

Функции безопасности базового уровня защищённости («Орел»):

- **Запрет вывода меню загрузчика** — при выборе данного пункта меню загрузчика GRUB2 не будет отображаться. Загрузка ядра ОС будет выполняться в соответствии со значением по умолчанию;
- **Запрет трассировки ptrace** — при выборе данного пункта будет отключена возможность трассировки и отладки выполнения программного кода;
- **Запрос пароля для команды sudo** — при выборе данного пункта будет включено требование ввода пароля при использовании механизма sudo;
- **Запрет установки бита исполнения** — при выборе данного пункта будет включен режим запрета установки бита исполнения, обеспечивающий предотвращение несанкционированного создания пользователями исполняемых сценариев для командной оболочки;
- **Запрет исполнения скриптов пользователя** — при выборе данного пункта будет заблокировано интерактивное использование пользователем интерпретаторов;
- **Запрет исполнения макросов пользователя** — при выборе данного пункта будет заблокировано исполнение макросов в стандартных приложениях;
- **Запрет консоли** — при выборе данного пункта пользователям будет заблокирован консольный вход в систему и запуск консоли из графической сессии пользователя;
- **Системные ограничения ulimits** — при выборе данного пункта будут включены системные ограничения;
- **Запрет автонастройки сети** — при выборе данного пункта будет отключена автоматическая настройка сети в процессе установки ОС, сеть необходимо будет настроить вручную;
- **Местное время для системных часов** — при выборе данного пункта системные часы будут установлены на местное время.

  Рекомендуется включить при совместной работе на компьютере с операционными системами семейства Windows.

Для усиленного уровня защищённости («Воронеж») доступны все функции безопасности базового уровня («Орел»), а также следующие:

- **Мандатный контроль целостности** — при выборе данного пункта будет включен мандатный контроль целостности;
- **Замкнутая программная среда** — при выборе данного пункта будет включен механизм, обеспечивающий проверку неизменности и подлинности загружаемых исполняемых файлов формата ELF;
- **Очистка освобождаемой внешней памяти** — при выборе данного пункта будет включен режим очистки блоков файловой системы непосредственно при их освобождении, а также режим очистки разделов страничного обмена.

Для максимального уровня защищённости («Смоленск») доступны все функции безопасности усиленного уровня («Воронеж»), а также **Мандатное управление доступом** — при выборе данного пункта будет включено мандатное управление доступом.