Операционные системы

Тормозов В.С.

Учебник

Публикуется в авторской редакции

2026

Владимир Тормозов

Операционные системы

https://litres.ru/74807852
SelfPub; 2026

Аннотация

В учебнике рассмотрены базовые понятия операционных систем, принципы их построения и функционирования, управление процессами, синхронизация, взаимоблокировки, межпроцессное взаимодействие и управление памятью.

Учебник содержит теоретический материал, практические примеры, вопросы для самоконтроля и тестовые задания по каждому разделу.

Учебник соответствует учебно-методическому комплексу дисциплины «Операционные системы» для направления подготовки 09.03.04 «Программная инженерия» (уровень бакалавриата), реализуемой в ФГБОУ ВО «Тихоокеанский государственный университет» (ТОГУ).

Предназначено для студентов направления подготовки 09.03.04 «Программная инженерия» (бакалавриат), изучающих дисциплину «Операционные системы» в ФГБОУ ВО «Тихоокеанский государственный университет», а также для всех, кто начинает знакомство с устройством и настройкой современных операционных систем.

Содержание

1. Основные понятия операционных систем 7
1.2 Типы операционных систем 11
2. Концепции операционных систем 32
2.1. Ядро операционной системы 32
2.2. Драйверы устройств 33
2.3. Концепция обратных вызовов 37
2.4. Режимы работы процессора 43
2.5 Системные вызовы операционной системы 49
2.6. Загрузчик операционной системы 53
3. Работа программного обеспечения в концепции операционных систем 54
3.1. Стандартные потоки программ 54
3.2. Интерфейс прикладного программирования (API) 56
Конец ознакомительного фрагмента. 57

Владимир Тормозов
Операционные системы

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Тормозов В. С.

Учебник
Публикуется в авторской редакции

ЛитРес : Самиздат
2026
УДК 004.42(075.8)
ББК 32.973-018.1я73
Т 60

Рецензенты:
Dr. Arvind Kumar Menon, PhD
Dr. Alok Kumar Sharma, PhD

Тормозов, В. С.
Операционные системы / В. С. Тормозов. – ЛитРес : Самиздат, 2026. – 292 с.

В учебнике рассмотрены базовые понятия операционных систем, принципы их построения и функционирования, управление процессами, синхронизация, взаимоблокировки, межпроцессное взаимодействие и управление памятью. Приведены сведения о типах операционных систем, ядре, драйверах, системных вызовах, пользовательских интерфейсах, методах управления памятью и виртуальной памяти, а также об установке и настройке Linux Ubuntu.

Учебник содержит теоретический материал, практические примеры, вопросы для самоконтроля и тестовые задания по каждому разделу.

Учебник соответствует учебно-методическому комплексу дисциплины «Операционные системы» для направления подготовки 09.03.04 «Программная инженерия» (уровень бакалавриата), реализуемой в ФГБОУ ВО «Тихоокеанский государственный университет» (ТОГУ).

Предназначено для студентов направления подготовки 09.03.04 «Программная инженерия» (бакалавриат), изучающих дисциплину «Операционные системы» в ФГБОУ ВО «Тихоокеанский государственный университет», а также для всех, кто начинает знакомство с устройством и настройкой современных операционных систем.

УДК 004.42(075.8)
ББК 32.973-018.1я73
© Тормозов В. С., 2026
© Оформление. ЛитРес : Самиздат, 2026

Посвящается моим учителям,
открывшим для меня мир операционных систем
и системного программирования.

1. Основные понятия операционных систем

.1. Определение операционной системы

Операционная система (ОС) представляет собой самую главную программу на любом компьютере, смартфоне, планшете или игровом устройстве. Без нее электроника является просто сложным набором микросхем, проводов, кнопок и экрана, который не способен выполнять понятные и полезные для человека задачи. В момент нажатия кнопки питания на корпусе сначала запускается базовое встроенное микропрограммное обеспечение, которое проверяет исправность деталей, а затем сразу передает управление операционной системе, загружая ее с накопителя в память. Чтобы разобраться в том, как устроена работа современного компьютера, все существующее программное обеспечение принято делить на два больших уровня.

К первому уровню относится системное программное обеспечение, главной частью которого является сама операционная система. Ее основная задача заключается в обслуживании самого устройства и постоянном контроле над слаженной работой всех его компонентов. Внутри любого компьютера существуют аппаратные ресурсы, то есть физиче-

ские детали, которые можно увидеть и потрогать руками, включая центральный процессор, планки оперативной памяти, жесткий диск или твердотельный накопитель, видеокарту, звуковой чип, монитор, клавиатуру и мышь. Помимо этого существуют программные ресурсы в виде запущенных фоновых процессов, служебных файлов, очередей задач и сетевых подключений. Операционная система выступает в роли строгого диспетчера, управляющего и переводчика, который принимает команды от человека, переводит их на понятный для электронных схем машинный язык из нулей и единиц и справедливо распределяет вычислительную нагрузку.

Ко второму уровню относятся прикладные программы, то есть все привычные приложения, которыми ежедневно пользуется человек для работы, учебы и развлечений, включая интернет-браузеры, текстовые редакторы, таблицы, мессенджеры, графические пакеты, плееры для музыки и видеоигры. Обычные прикладные программы не умеют и не имеют права общаться с физическим оборудованием компьютера напрямую, поскольку каждая модель процессора, видеокарты или принтера устроена по своим правилам.

Чтобы разработчикам приложений не приходилось писать отдельный код под тысячи разных моделей деталей, в операционной системе предусмотрены специальные программы переводчики, которые называются драйверами. Например, при нажатии кнопки сохранения в текстовом редакторе про-

грамма совершенно не знает, на какой именно физический сектор жесткого диска нужно записать документ, поэтому она просто передает готовый текст операционной системе, а та уже через драйверы сама находит свободное место на диске. Вся подобная рутинная техническая работа ложится на общие службы, которые ОС предоставляет всем установленным программам.

Первой такой службой является управление процессором и оперативной памятью, когда при одновременном запуске браузера с десятком вкладок, музыкального плеера и антивируса система по тысяче раз в секунду переключает внимание процессора между задачами и выделяет каждой ровно столько памяти, сколько нужно для плавной работы без зависаний.

Второй важной функцией выступает работа с устройствами ввода и вывода, при которой ОС мгновенно считывает движения компьютерной мыши, нажатия клавиш или прикосновения к сенсорному экрану, определяет, в какое именно активное окно направить этот сигнал, а также отвечает за формирование картинки на мониторе и передачу звука в колонки или наушники.

Третьей службой считается файловая система, организующая хранение всей цифровой информации в виде привычной для человека иерархии из папок и файлов, при этом система строго следит за тем, чтобы новые данные случайно не записались поверх старых и не повредили их.

Четвертой важной задачей является сетевое взаимодействие, благодаря которому приложения могут отправлять и получать данные через кабель или беспроводную сеть, не задумываясь о том, как именно устроена передача радиосигнала.

Пятой критической службой выступает обеспечение безопасности и разграничение прав доступа, когда система защищает важные служебные файлы от случайного удаления пользователем, проверяет учетные записи и изолирует программы друг от друга, чтобы ошибка или вирус в одном приложении не смогли сломать весь компьютер. Кроме того, операционная система предоставляет человеку удобный пользовательский интерфейс в виде рабочего стола с окнами, значками и меню, либо в виде текстовой командной строки для опытных специалистов. Самыми распространенными примерами операционных систем для настольных компьютеров и ноутбуков выступают Windows, macOS и Linux, а для смартфонов и планшетов используются Android и iOS.

Операционная система (ОС) – это системное программное обеспечение, которое управляет аппаратными и программными ресурсами компьютера и предоставляет общие службы для компьютерных программ. Прикладные программы обычно требуют наличия операционной системы для своей работы.

1.2 Типы операционных систем

Операционные системы бывают очень разными, поскольку задачи у современных электронных устройств сильно отличаются друг от друга и требуют совершенно непохожих подходов к управлению ресурсами. Обычному домашнему компьютеру или офисному ноутбуку в первую очередь нужна красивая и понятная графическая картинка, поддержка огромного количества разнообразных программ и максимальное удобство для одного конкретного человека, который сидит перед монитором с клавиатурой и мышью. Мощному серверу в большом центре обработки данных, напротив, абсолютно не нужны красочные анимации и визуальные эффекты, зато критически важна бесперебойная круглосуточная работа месяцами без перезагрузки и способность одновременно обслуживать тысячи запросов из интернета без малейших задержек и сбоев. Карманному смартфону, планшету или умным часам требуется совершенно другой баланс качеств, где на первое место выходит умение бережно расходовать ограниченный заряд аккумулятора, мгновенно реагировать на прикосновения пальцев к сенсорному экрану, управлять камерами, датчиками положения в пространстве и постоянно поддерживать устойчивую связь с сотовыми вышками и беспроводными сетями.

Совсем иные и максимально строгие требования предъяв-

ляются к управляющей электронике внутри современных автомобилей, сложных медицинских аппаратов жизнеобеспечения, промышленных станков, систем управления воздушным движением или космических зондов, где от точности и скорости реакции на показания датчиков напрямую зависят жизни людей, а задержка даже на долю секунды способна привести к серьезной аварии.

Помимо этого в повседневной жизни нас окружает огромное количество невидимых на первый взгляд программных оболочек, которые незаметно для глаз работают внутри привычной бытовой техники, стиральных машин, умных телевизоров, домашних роутеров, кассовых аппаратов и банковских терминалов, выполняя строго ограниченный набор заложенных на заводе функций при минимальных затратах памяти.

Наконец, в крупных научных лабораториях и корпорациях применяются сложные программные комплексы, способные объединять десятки и сотни отдельных вычислительных машин в единую слаженную сеть так, чтобы для исследователей и запущенных расчетов они выглядели как один гигантский суперкомпьютер с общими файлами и колоссальной мощностью. Из за таких противоположных условий эксплуатации одна универсальная программа просто не смогла бы одинаково хорошо и безопасно работать на всех существующих приборах сразу.

Попытка установить тяжелую систему с обычного домаш-

него компьютера на маленький контроллер умной лампочки или на бортовой компьютер самолета привела бы к мгновенному переполнению памяти, зависаниям и полному отказу оборудования. Именно поэтому инженеры и программисты проектируют операционные системы с разной внутренней архитектурой, различными алгоритмами распределения процессорного времени и оперативной памяти, разными уровнями защиты от сбоев и индивидуальными правилами взаимодействия с оборудованием. Одни системы строятся вокруг максимальной скорости отклика на внешние сигналы, другие делают ставку на абсолютную надежность и защиту данных от постороннего вмешательства, а третьи жертвуют скоростью ради плавности интерфейса и простоты освоения обычным пользователем без специальных технических знаний.

Также системы сильно различаются по тому, как именно они организуют выполнение нескольких дел одновременно и сколько человек могут работать с устройством в один и тот же момент времени. В одних случаях устройство жестко закреплено за одним владельцем, а в других к одной мощной машине по сети могут одновременно подключаться сотни сотрудников, и система обязана справедливо делить между ними вычислительные ресурсы, строго следя за тем, чтобы действия одного человека никак не мешали работе другого.

Чтобы специалистам, разработчикам и обычным пользо-

вателям было проще ориентироваться в этом огромном многообразии программных платформ, всю совокупность существующих операционных систем принято делить на отдельные категории и классы по их главному назначению, особенностям аппаратной части, количеству одновременно обслуживаемых пользователей, способу распределения вычислительной мощности и строгости требований к скорости реакции на внешние события.

Чтобы легко разобраться в дальнейшем перечислении классов операционных систем, важно понимать сам принцип, по которому авторы разделяют все программы на группы. Любая классификация в информатике устроена точно так же, как сортировка книг в большой библиотеке или товаров на полках магазина, где одни и те же предметы можно разложить по разным признакам в зависимости от цели.

Одну и ту же операционную систему можно одновременно отнести сразу к нескольким разным группам, просто взглянув на нее с разных сторон, например со стороны ее внешнего вида, количества людей за экраном или скорости реакции на команды. Когда вы будете читать список категорий дальше по учебнику, обращайте внимание на то, какой именно главный вопрос лежит в основе каждого пункта.

Если речь идет о том, где именно стоит устройство и для чего оно создано, то перед вами разделение по области применения, которое отделяет привычные домашние компьютеры и смартфоны от невидимых систем внутри бытовой тех-

ники или мощных машин в дата центрах. Если же в описании говорится о том, сколько дел система умеет делать за одну секунду, то авторы объясняют разницу между простыми устаревшими программами, способными выполнять только одну операцию по очереди, и современными системами, которые умеют мгновенно переключаться между десятками открытых окон и фоновых процессов.

Другой важный признак касается количества человек, которым разрешено одновременно отдавать команды одному устройству, что позволяет отделить личный ноутбук одного владельца от сложного рабочего узла, к которому по сети подключаются сотни сотрудников со своими паролями и личными папками.

Отдельное внимание в классификации всегда уделяется времени отклика, поскольку для обычного пользователя задержка в половину секунды при открытии папки остается совершенно незаметной, а для автоматики на заводе или в автомобиле такая пауза считается критической ошибкой. Кроме того, системы часто делят по способу общения с человеком на те, где нужно вводить текстовые команды с клавиатуры, и те, где управление происходит через привычные значки, окна и нажатия пальцем по экрану, а также по юридическому статусу и открытости исходного кода, что определяет, может ли любой программист бесплатно изучать и менять внутреннее устройство системы или она полностью закрыта и принадлежит конкретной коммерческой компании. Пони-

мание этих простых логических критериев поможет вам не просто зазубрить длинный список терминов из учебника, а сразу видеть за каждым названием конкретные технические возможности и реальные жизненные задачи, ради которых эта система была создана.

1.3 Однозадачные и многозадачные операционные системы

Однозадачная система может запускать только одну программу одновременно, в то время как многозадачная операционная система позволяет одновременно запускать более одной программы. Это достигается за счет разделения времени, когда доступное процессорное время распределяется между несколькими процессами, каждый из которых многократно прерывается по временным интервалам подсистемой планирования задач операционной системы. Многозадачность может быть разделена на вытесняющую и кооперативную. При вытесняющей многозадачности операционная система делит процессорное время на интервалы и выделяет слот каждой из программ. Unix-подобные операционные системы, например, Solaris, Linux, а также AmigaOS, поддерживают вытесняющую многозадачность. Кооперативная многозадачность достигается за счет того, что каждый процесс предоставляет время другим процессам определенным образом. 16-битные версии Microsoft Windows использовали кооперативную многозадачность. 32-битные версии Windows NT и Win9x использовали вытесняющую многоза-

дачность. С другой стороны, важно различать многозадачную операционную систему и многозадачный пользовательский интерфейс. Например, iOS основана на Unix и, следовательно, является многозадачной операционной системой (сотовые службы, пользовательский интерфейс, музыкальный плеер – все это процессы, работающие одновременно), но ее интерфейс не позволял пользователю запускать более одного приложения одновременно до версии 4.0.

1.4 Однопользовательские и многопользовательские операционные системы

Однопользовательские операционные системы не имеют средств для различения пользователей, но могут позволять одновременное выполнение нескольких программ. Многопользовательская операционная система расширяет базовую концепцию многозадачности, добавляя средства, позволяющие идентифицировать процессы и ресурсы, такие как дисковое пространство, принадлежащие нескольким пользователям, и позволяет нескольким пользователям взаимодействовать с системой одновременно. Операционные системы с разделением времени планируют задачи для эффективного использования системы и могут также включать программное обеспечение для учета затрат на распределение процессорного времени, памяти, печати и других ресурсов между несколькими пользователями. Однопользовательские операционные системы спроектированы таким образом, что в их архитектуре полностью отсутствуют программные средства

и механизмы, предназначенные для различения и разделения нескольких пользователей.

Для подобной программной среды любой человек, осуществляющий взаимодействие с вычислительным устройством, воспринимается как единый и единственный субъект без разграничения прав или выделения изолированных рабочих пространств. Вместе с тем отсутствие встроенных инструментов идентификации отдельных лиц никоим образом не ограничивает функциональные возможности системы в плане организации параллельных вычислений. Подобные операционные системы вполне могут поддерживать режим многозадачности, позволяя одному пользователю инициировать и обеспечивать одновременное выполнение сразу нескольких различных прикладных программ.

Многопользовательская операционная система берет за основу эту базовую концепцию многозадачности и существенно расширяет ее функциональные границы. Данное расширение реализуется за счет внедрения специализированных системных средств, которые позволяют программному обеспечению безошибочно идентифицировать каждый запущенный вычислительный процесс и точно определять его принадлежность конкретному человеку. Аналогичным образом система отслеживает и разграничивает все выделяемые аппаратные и программные ресурсы, среди которых в качестве наглядного примера можно выделить дисковое пространство, закрепленное за разными людьми.

Благодаря наличию таких строгих механизмов привязки процессов и ресурсов к конкретным учетным субъектам, многопользовательская среда позволяет нескольким пользователям полноценно взаимодействовать с одной и той же вычислительной системой одновременно. При этом каждый участник работы получает возможность выполнять собственные задачи параллельно с остальными, а система гарантирует корректное разделение их данных и вычислительных потоков.

Логическим развитием данного подхода выступают операционные системы с разделением времени, главной задачей которых является грамотное и последовательное планирование поступающих задач. Подобное планирование осуществляется с целью достижения максимально эффективного использования всех вычислительных мощностей и аппаратных возможностей системы в условиях высокой нагрузки.

Помимо непосредственного составления расписания выполнения процессов, архитектура операционных систем с разделением времени может также включать в свой состав специализированное программное обеспечение, предназначенное для детального учета затрат. Данный учетный инструментарий фиксирует и подсчитывает точный объем потребления различных системных ресурсов между несколькими пользователями, включая затраченное процессорное время, выделенный объем оперативной памяти, использование устройств печати и прочих общих возможностей вычис-

лительного комплекса.

1.5 Распределённые операционные системы

Распределенная операционная система управляет группой отдельных компьютеров, создавая впечатление, что это один компьютер. ^ Развитие сетевых компьютеров, которые могли быть связаны и взаимодействовать друг с другом, привело к появлению распределенных вычислений. Распределенные вычисления выполняются на нескольких машинах. Когда компьютеры в группе работают совместно, они образуют распределенную систему. Распределенная операционная система представляет собой специализированную системную программную среду, ключевое предназначение которой заключается в том, чтобы брать на себя комплексное и централизованное управление целой группой отдельных, физически независимых компьютеров.

В рамках этой концепции множество самостоятельных вычислительных устройств объединяется в единую скоординированную архитектуру, при этом программная оболочка организована таким образом, что она полностью скрывает от наблюдателя сложную структуру взаимодействия множества узлов.

Главная функциональная особенность подобного системного подхода состоит в том, что распределенная операционная система виртуализирует и объединяет аппаратные возможности всех разрозненных машин, целенаправленно создавая для конечного пользователя или запущенной при-

кладной программы полное и неотличимое впечатление, что перед ними функционирует один единый, целостный и монолитный компьютер. Вся физическая разобщенность отдельных вычислительных блоков, их территориальная удаленность друг от друга и необходимость постоянной пересылки данных между ними остаются абсолютно невидимыми на уровне пользовательского восприятия, формируя иллюзию взаимодействия с единым вычислительным ресурсом.

Своим историческим появлением и последующим активным распространением данная программная концепция напрямую обязана стремительному технологическому развитию сетевых компьютеров в области вычислительной техники. По мере того как отдельные электронные вычислительные машины обретали техническую возможность быть надежно связанными друг с другом посредством коммуникационных каналов и начинали беспрепятственно взаимодействовать между собой путем регулярного обмена информацией и управляющими сигналами, в компьютерных науках и инженерной практике закономерно сформировалось принципиально новое направление, получившее название распределенных вычислений.

Сама фундаментальная суть распределенных вычислений сводится к отказу от локализации обработки информации в пределах одного замкнутого корпуса. Вместо того чтобы пытаться решить сложную вычислительную задачу или выполнить объемный программный код исключительно си-

лами одного изолированного устройства, вычислительный процесс целенаправленно разделяется и выполняется сразу на нескольких отдельных машинах одновременно.

Каждое из вовлеченных в этот процесс устройств берет на себя свою часть общей работы, обрабатывая данные параллельно с другими участниками вычислительного комплекса. В тот самый момент, когда отдельные компьютеры, входящие в состав общей объединенной группы, переходят от автономного функционирования к слаженным действиям и начинают систематически работать совместно ради достижения единых вычислительных целей, они качественно меняют свой статус. Переставая быть просто механическим набором изолированных и независимых друг от друга электронных устройств, эти машины в совокупности образуют качественно новую структуру, которая в науке и технике определяется как единая распределенная система.

Таким образом, именно распределенная операционная система выступает тем самым ключевым управляющим и связующим программным механизмом, который организует, координирует и поддерживает это сложное совместное взаимодействие множества самостоятельных машин в рамках единой распределенной системы. Она берет на себя всю координацию параллельных процессов, обеспечивая согласованную работу группы компьютеров так, чтобы распределенные вычисления проходили слаженно, а внешнее восприятие всей этой сложной сетевой инфраструктуры неизменно

сводилось к представлению о работе на одном единственном компьютере.

1.6 Встраиваемые операционные системы

Встраиваемые операционные системы предназначены для использования во встраиваемых компьютерных системах. Они разработаны для работы на небольших устройствах, таких как КПК, умные часы или другие устройства с меньшей автономностью. Они способны работать с ограниченным количеством ресурсов. По своей конструкции они очень компактны и чрезвычайно эффективны. watchOS, Windows CE, Minix 3 – это некоторые примеры встраиваемых операционных систем. Встраиваемые операционные системы представляют собой особый, узкоспециализированный класс системного программного обеспечения, который изначально проектируется и целенаправленно предназначается для непосредственного использования исключительно во встраиваемых компьютерных системах.

В отличие от привычных универсальных программных платформ, рассчитанных на полноразмерные настольные персональные компьютеры, производительные рабочие станции или мощные вычислительные серверы, данный вид систем создается в строгом соответствии с жесткими физическими, энергетическими и техническими рамками целевого электронного оборудования. Они изначально разработаны инженерами и программистами для обеспечения стабильной, предсказуемой и абсолютно бесперебойной работы

на компактных, миниатюрных и небольших по своим габаритам устройствах.

В качестве наиболее характерных и наглядных примеров подобных приборов можно выделить классические карманные персональные компьютеры (КПК), современные наручные умные часы, а также множество других портативных, носимых или специализированных электронных устройств, которые по своей природе отличаются меньшей степенью автономности. Специфика конструкции и повседневной эксплуатации подобных миниатюрных гаджетов и специализированных приборов неизбежно накладывает весьма серьезные и жесткие ограничения на их внутреннее аппаратное оснащение. Из за малых размеров корпуса и строгих требований к энергопотреблению производители не могут разместить внутри таких приборов мощные многоядерные процессоры или модули памяти большого объема, поэтому ключевым и определяющим требованием к подобным программным платформам становится умение полноценно и безотказно функционировать в условиях острой нехватки вычислительных мощностей.

Встраиваемые операционные системы изначально сконструированы так, чтобы быть способными уверенно и эффективно работать с весьма ограниченным количеством системных ресурсов, не требуя для своего запуска и выполнения задач значительных аппаратных резервов.

Для того чтобы успешно справляться со всеми постав-

ленными функциональными задачами в столь стесненных технических условиях, не перегружать скромную аппаратную часть и не вызывать замедлений в работе прибора, эти программные решения подвергаются глубочайшей оптимизации на самом фундаментальном архитектурном уровне. Разработчики сознательно исключают из их состава любые второстепенные, громоздкие или избыточные компоненты, оставляя исключительно те программные механизмы, которые жизненно необходимы для обслуживания конкретного устройства.

По своей внутренней программной конструкции, логической организации кода и архитектуре ядра встраиваемые операционные системы являются очень компактными, занимая минимально возможный объем в памяти устройства. При всей своей миниатюрности и внешней простоте они одновременно с этим демонстрируют чрезвычайную эффективность при выполнении заложенных в них управляющих и вычислительных операций, расходуя каждый такт процессора и каждый байт памяти с максимальной пользой. Благодаря такому тщательно выверенному инженерному балансу между предельно малым занимаемым объемом и высокой результативностью выполнения команд, подобные системы обеспечивают стабильное управление небольшими устройствами с меньшей автономностью без лишних затрат вычислительных циклов. Чтобы закрепить теоретическое представление о данном классе программного обеспечения и свя-

зать его с реальной практикой разработки, можно обратиться к конкретным и широко известным реализациям.

В качестве наиболее показательных примеров, наглядно иллюстрирующих практическое применение встраиваемых операционных систем в различных электронных устройствах, выступают такие программные продукты, как операционная система watchOS, платформа Windows CE, а также система Minix 3, каждая из которых наглядно подтверждает принципы компактности и высокой эффективности при работе на оборудовании с ограниченными ресурсами.

1.7 Операционная система реального времени

Операционная система реального времени – это операционная система, которая гарантирует обработку событий или данных в течение определенного короткого промежутка времени. Операционная система реального времени может быть однозадачной или многозадачной, но при многозадачности она использует специализированные алгоритмы планирования, что обеспечивает детерминированный характер поведения.

Событийно-ориентированная система переключается между задачами на основе их приоритетов или внешних событий, в то время как операционные системы с разделением времени переключают задачи на основе прерываний от часов. Операционная система реального времени представляет собой специализированный класс системного программного обеспечения, ключевой и определяющей характеристи-

кой которого является строгая гарантия обработки поступающих событий или данных в течение строго определенного, заранее заданного короткого промежутка времени.

В отличие от привычных программных платформ общего назначения, где время отклика может колебаться в зависимости от текущей нагрузки на вычислительные мощности, операционная система реального времени строится вокруг безусловного соблюдения временных рамок и своевременного реагирования на внешние сигналы. Главная задача подобной системы состоит в том, чтобы гарантировать завершение обработки любого события или массива данных строго в пределах установленного короткого временного интервала без каких бы то ни было непредсказуемых задержек.

По своей внутренней организации и функциональным возможностям операционная система реального времени может быть реализована как в однозадачном варианте, когда вычислительные ресурсы последовательно обслуживают одну текущую операцию, так и в многозадачном исполнении, позволяющем одновременно управлять множеством параллельных процессов. Однако в том случае, когда операционная система реального времени функционирует в режиме многозадачности, она принципиально отказывается от стандартных механизмов распределения ресурсов и использует исключительно специализированные алгоритмы планирования.

Применение таких специализированных алгоритмов пла-

нирования является критически важным архитектурным решением, поскольку именно оно обеспечивает строго детерминированный характер поведения всей программной среды. Детерминированный характер поведения означает полную предсказуемость, повторяемость и строгую определенность работы системы во времени, благодаря чему заранее известно, как именно и в какие конкретные временные рамки система отреагирует на каждую задачу при многозадачности. Для понимания внутренних механизмов смены выполняемых операций в подобных платформах необходимо различать принципы их переключения.

В частности, событийно ориентированная система организована таким образом, что она оперативно переключается между различными активными задачами на основе их заранее присвоенных приоритетов или при непосредственном поступлении внешних событий, требующих немедленного внимания. Если возникает внешнее событие или появляется задача с более высоким уровнем приоритета, событийно ориентированная система прерывает менее срочную работу и переключается на выполнение более важной операции, обеспечивая тем самым гарантированную обработку в течение короткого промежутка времени.

Совершенно иной подход к управлению очередью процессов демонстрируют операционные системы с разделением времени, которые строят свою работу без жесткой привязки к срочности поступающих сигналов. В отличие от со-

бытийно ориентированных платформ, операционные системы с разделением времени переключают задачи планомерно и циклично, опираясь исключительно на регулярные прерывания от системных часов.

Пока операционные системы с разделением времени равномерно делят процессорные циклы между процессами по сигналу внутренних часов, операционная система реального времени за счет использования специализированных алгоритмов планирования в многозадачном режиме и опоры на приоритеты или внешние события в событийно ориентированной архитектуре сохраняет свой детерминированный характер поведения и неукоснительно гарантирует обработку любых событий или данных в течение строго определенного короткого промежутка времени.

1.8 Основные операционные системы

В рамках изучения компьютерных технологий и системного программного обеспечения особое внимание уделяется ознакомлению с ключевыми программными платформами, которые составляют основу современной вычислительной техники и определяются как основные операционные системы. Когда речь заходит о систематизации наиболее значимых, широко распространенных и фундаментальных решений в данной области, в первую очередь выделяют ряд конкретных программных продуктов, каждый из которых занимает свое важное место в общем перечне основных операционных систем. Открывает этот базовый перечень опе-

рационная система Linux, которая представляет собой не просто одну изолированную платформу, а целое семейство, включающее в свой состав такие известные и распространенные дистрибутивы, как Debian, Ubuntu, Fedora, CentOS и SuSE. Каждый из перечисленных вариантов является полноценным дистрибутивом в рамках общей экосистемы Linux и неотъемлемой частью списка основных операционных систем.

Далее в ряду основных операционных систем следует платформа Android, представляющая собой самостоятельную и широко известную системную среду в общем перечне ключевых решений. Помимо нее, в фундаментальный список основных операционных систем неизменно входит семейство BSD, которое представляет важное направление в развитии системного программного обеспечения. Следующей значимой позицией в перечне основных операционных систем выступает платформа macOS, занимающая свое обособленное место среди базовых программных оболочек. Вслед за ней в списке основных операционных систем закономерно выделяется платформа iOS, также являющаяся неотъемлемым элементом общей классификации ключевых системных продуктов. Отдельным самостоятельным пунктом в перечне основных операционных систем упоминается система FreeBSD, которая дополняет представление о разнообразии существующих программных решений наряду с общей ветвью BSD. Продолжает этот перечень операционная

система Солярис, представляющая собой еще один важный и самостоятельный пример в ряду основных операционных систем. Завершает данный фундаментальный список операционная система Windows, которая является одной из наиболее известных платформ в общей совокупности основных операционных систем. Таким образом, полный перечень основных операционных систем последовательно охватывает семейство Linux, включая такие его конкретные дистрибутивы, как Debian, Ubuntu, Fedora, CentOS и SuSE, а также платформы Android, BSD, macOS, iOS, FreeBSD, Солярис и Windows, формируя целостное представление о базовых программных средах.

2. Концепции операционных систем

2.1. Ядро операционной системы

С помощью встроенного ПО и драйверов устройств ядро обеспечивает базовый уровень управления всеми аппаратными устройствами компьютера. Оно управляет доступом к памяти для программ в ОЗУ, определяет, какие программы получают доступ к каким аппаратным ресурсам, устанавливает или сбрасывает рабочие состояния ЦП для оптимальной работы и организует данные для долговременного энергонезависимого хранения с помощью файловых систем на таких носителях, как диски, ленты, флэш-память и т. д .

2.2. Драйверы устройств

Драйвер устройства – это программное обеспечение, которое сообщает операционной системе, как взаимодействовать с устройством. Предоставляя программный интерфейс для аппаратных устройств, драйвер позволяет операционным системам и другим компьютерным программам получать доступ к функциям оборудования, не требуя знания точных характеристик используемого оборудования. Драйвер устройства представляет собой специализированное системное программное обеспечение, основное и фундаментальное функциональное предназначение которого заключается в том, чтобы детально, последовательно и точно сообщать операционной системе, как именно необходимо взаимодействовать с конкретным подключенным аппаратным устройством. В сложной многоуровневой архитектуре современной вычислительной техники именно драйвер устройства выступает тем абсолютно необходимым связующим компонентом и программным посредником, который берет на себя сложную задачу обеспечения корректного, бесперебойного и безошибочного диалога между базовой программной средой и физическим оборудованием. Если бы данное специализированное программное обеспечение отсутствовало в системе, операционная система попросту не располагала бы необходимыми инструкциями и сведениями о том,

каким образом передавать команды конкретному устройству и как правильно интерпретировать поступающие от него ответные сигналы.

Главный принцип функционирования данного программного механизма реализуется за счет того, что драйвер разрабатывается как слой программной абстракции и предоставляет четкий, унифицированный программный интерфейс для различных аппаратных устройств. Благодаря целенаправленному созданию и последующему предоставлению этого стандартизированного программного интерфейса для аппаратных устройств, драйвер эффективно скрывает и изолирует всю внутреннюю техническую сложность и специфику работы физического оборудования от всех вышестоящих уровней программного обеспечения.

Наличие подобного единого программного интерфейса означает, что вместо необходимости разбираться в уникальных внутренних механизмах функционирования каждого отдельно взятого компонента, программная среда получает готовый, понятный и стандартизированный набор программных инструментов для обращения к оборудованию.

В результате внедрения такого выверенного архитектурного решения драйвер позволяет как самим операционным системам, так и другим разнообразным компьютерным программам беспрепятственно, надежно и полноценно получать доступ ко всем необходимым функциям оборудования. Вся функциональная мощь и все заложенные производителем

возможности физического компонента становятся открытыми для использования со стороны программного обеспечения исключительно благодаря тому, что драйвер выступает переводчиком команд через свой программный интерфейс. Примечательной, определяющей и критически важной особенностью всего этого процесса является то обстоятельство, что операционные системы и другие компьютерные программы получают этот полноценный доступ к функциям оборудования, совершенно не требуя знания точных технических характеристик используемого оборудования.

Программной среде и многочисленным прикладным компьютерным программам нет абсолютно никакой необходимости владеть детальной информацией о внутренних спецификациях, особенностях конструкции, аппаратных протоколах или точных параметрах конкретного используемого оборудования.

Поскольку всю сложную работу по учету этих низкоуровневых деталей берет на себя специализированное программное обеспечение в лице драйвера, операционные системы и другие компьютерные программы могут задействовать любые функции оборудования в унифицированном порядке, полностью освобождаясь от потребности знать точные характеристики используемого оборудования. При этом каждый раз, когда возникает необходимость обратиться к аппаратной части, компьютерная программа или сама операционная система использует предоставляемый программный

интерфейс, а драйвер устройства трансформирует эти стандартизированные запросы в конкретные действия, понятные данному устройству.

Вся логика взаимодействия с устройством остается сосредоточенной исключительно внутри этого специализированного программного обеспечения, что избавляет операционную систему и иные компьютерные программы от необходимости содержать в себе сведения о точных характеристиках используемого оборудования и позволяет им сосредоточиться на выполнении своих непосредственных задач, опираясь на стабильный доступ к функциям оборудования.

1.11. Встроенное программное обеспечение

Встроенное программное обеспечение (прошивка) – это тип программного обеспечения, обеспечивающий управление, мониторинг и обработку данных инженерных изделий и систем, навсегда заложенный в аппаратное обеспечение. Типичными примерами устройств, содержащих прошивку, являются встроенные системы (такие как светофоры, бытовая техника и цифровые часы), компьютеры, компьютерная периферия, мобильные телефоны и цифровые камеры, но не все устройства имеют прошивку (только устройства с определенным *уровнем интеллекта*). Прошивка, содержащаяся в этих устройствах, обеспечивает низкоуровневую программу управления устройством. По состоянию на 2013 год большинство прошивок можно обновить.

2.3. Концепция обратных вызовов

Функция обратного вызова – это фрагмент исполняемого кода, который передается в качестве аргумента другому коду, который, как ожидается, вызовет (выполнит) этот аргумент в удобное для него время.

```
void error_func(const char* msg)
{
printf("ERROR: %sn", msg);
}

float divide(float a, float b, void(*error_callback)(const char*))
{
if (b == 0.0f) {
error_callback("Divisor can't be zero");

return 0.0f;
}

return a / b;
}

// ...
float c = divide(15.0f, 3.0f, error_func);
```

```
float d = divide(c, 0.0f, error_func);
// ...
```

1.13 Концепция прерывания и исключения

Прерывание – это сигнал, посылаемый процессору аппаратным или программным обеспечением, указывающий на событие, требующее немедленного внимания. Прерывание оповещает процессор о высокоприоритетном состоянии, требующем прерывания текущего выполняемого процессором кода. В ответ процессор приостанавливает свою текущую деятельность, сохраняет свое состояние и выполняет функцию, называемую обработчиком прерываний (или подпрограммой обработки прерываний, ISR), для обработки события. Это прерывание является временным, и после завершения работы обработчика прерываний процессор возобновляет нормальную работу. Существует два типа прерываний: аппаратные и программные.

Аппаратные прерывания используются устройствами для передачи информации о необходимости внимания со стороны операционной системы. Внутри системы аппаратные прерывания реализуются с помощью электронных сигналов оповещения, которые отправляются процессору от внешнего устройства, являющегося либо частью самого компьютера, например, контроллера диска, либо внешним периферийным устройством.

Программное прерывание возникает либо из-за исключительной ситуации в самом процессоре (также называемой ло-

вушкой или исключением), либо из-за специальной инструкции в наборе инструкций, которая вызывает прерывание при выполнении.

1.14. Работа операционной системы с использованием таймера

Операционная система должна уметь планировать выполнение действий на будущее. Необходим механизм, позволяющий планировать выполнение действий в относительно точное время. Любой микропроцессор, желающий поддерживать операционную систему, должен иметь программируемый интервальный таймер, который периодически прерывает работу процессора. Программные таймеры могут использоваться процессом для планирования выполнения блока инструкций (обычно это функция обратного вызова). Например, игровой движок может установить таймер для определения окончания действия волшебного зелья, или почтовый клиент может запланировать проверку новых писем каждые 5 минут. Работа операционной системы с использованием таймера представляет собой фундаментальный механизм организации вычислительных процессов во времени, без которого невозможно представить функционирование современной программной среды.

Для обеспечения стабильной, предсказуемой и скоординированной работы всех компонентов вычислительного комплекса операционная система должна в обязательном порядке уметь планировать выполнение различных действий

на будущее. В сложной программной архитектуре постоянно возникает потребность отложить ту или иную операцию на определенный срок или задать строгую периодичность ее повторения, поэтому системе необходим надежный и точный механизм, позволяющий планировать выполнение действий в относительно точное время.

Без наличия подобного инструментария, способного отмерять временные интервалы и инициировать задачи в заданный момент, программная среда утратила бы контроль над своевременным исполнением отложенных операций. На аппаратном уровне реализация этой важнейшей функции предъявляет строгие требования к архитектуре центрального вычислительного элемента, в соответствии с которыми любой микропроцессор, желающий поддерживать операционную систему, должен обязательно иметь в своем составе программируемый интервальный таймер.

Основное назначение данного аппаратного компонента заключается в том, что этот программируемый интервальный таймер периодически прерывает работу процессора через заданные промежутки времени, посылая ему соответствующие управляющие сигналы. Именно благодаря этим регулярным прерываниям, которые генерирует программируемый интервальный таймер в микропроцессоре, операционная система получает возможность отслеживать ход времени, возвращать себе управление и точно планировать выполнение действий на будущее в относительно точное время.

Опираясь на эту аппаратную основу, система предоставляет прикладным задачам более высокий уровень абстракции в виде программных таймеров.

Программные таймеры могут активно использоваться любым запущенным процессом для того, чтобы заранее планировать выполнение определенного блока инструкций в нужный момент времени. В практике программирования и системной разработки таким отложенным блоком инструкций, который запускается по срабатыванию программного таймера, обычно выступает специализированная функция обратного вызова, ожидающая своего часа для исполнения. Чтобы наглядно проиллюстрировать практическую значимость и принцип применения данного механизма в повседневных компьютерных программах, можно обратиться к конкретным примерам из разных областей разработки.

Например, в индустрии интерактивных развлечений игровой движок может установить специальный программный таймер для точного определения момента окончания действия волшебного зелья в виртуальном мире, чтобы по истечении заданного срока выполнить соответствующую функцию обратного вызова и вернуть характеристики персонажа к исходному состоянию.

Другим показательным примером использования данного механизма в офисных и коммуникационных приложениях является ситуация, когда почтовый клиент применяет программный таймер, чтобы заранее запланировать регулярную

автоматическую проверку наличия новых писем каждые 5 минут, инициируя соответствующий блок инструкций строго в заданное относительно точное время без необходимости постоянного участия со стороны пользователя.

2.4. Режимы работы процессора

Современные процессоры поддерживают несколько режимов работы. Процессоры с такой возможностью используют как минимум два режима: защищенный режим и режим супервизора. Режим супервизора используется ядром операционной системы для низкоуровневых задач, требующих неограниченного доступа к оборудованию, таких как управление записью и удалением памяти, а также связь с такими устройствами, как видеокарты. Защищенный режим, напротив, используется практически для всего остального. Современные процессоры обладают развитой архитектурой и поддерживают сразу несколько различных режимов работы, что является фундаментальной основой для обеспечения стабильности и безопасности всей вычислительной системы.

Те процессоры, которые наделены подобной аппаратной возможностью разграничения привилегий, в обязательном порядке используют в своей повседневной работе как минимум два базовых режима функционирования, а именно защищенный режим и режим супервизора. Четкое разделение вычислительного процесса между защищенным режимом и режимом супервизора позволяет аппаратно разделить критически важные системные операции и обычные пользовательские действия, не допуская хаотичного вмешательства в работу электроники.

В этой двухуровневой модели режим супервизора целенаправленно используется исключительно ядром операционной системы для выполнения наиболее ответственных низкоуровневых задач. Главная особенность таких низкоуровневых задач заключается в том, что они требуют абсолютно неограниченного доступа ко всему подключенному оборудованию вычислительной машины.

Поскольку ядро операционной системы отвечает за жизнеспособность всей программной среды, ему необходимы максимальные полномочия в рамках режима супервизора для прямого взаимодействия с физической частью компьютера. В качестве наиболее показательных примеров подобных низкоуровневых задач, выполняемых ядром операционной системы в режиме супервизора при наличии неограниченного доступа к оборудованию, можно выделить прямое управление процессами записи и удаления памяти, а также непосредственную связь с такими сложными периферийными и графическими устройствами, как видеокарты.

Защищенный режим, напротив, представляет собой полную противоположность по уровню предоставляемых привилегий и используется практически для всего остального, что происходит в вычислительной системе за пределами работы ядра. В то время как ядро операционной системы решает фундаментальные вопросы управления записью и удалением памяти или взаимодействия с видеокартами в режиме супервизора, все остальные программные процессы функци-

онируют исключительно в рамках защищенного режима, где их прямой доступ к аппаратным ресурсам строго ограничен архитектурой современных процессоров.

Приложения работают в защищенном режиме и могут использовать оборудование только посредством связи с ядром, которое управляет всем в режиме супервизора. В защищенном режиме программы могут иметь доступ к более ограниченному набору инструкций процессора. В некоторых процессорах эти режимы также называются *кольцевыми.* Прикладные приложения в современной вычислительной архитектуре всегда работают исключительно в рамках защищенного режима и принципиально не обладают полномочиями для самостоятельного прямого обращения к аппаратной части компьютера.

Поскольку приложения работают в защищенном режиме, они могут использовать подключенное физическое оборудование исключительно посредством организованной связи с ядром операционной системы. Именно ядро выступает тем единственным привилегированным компонентом, которое управляет всем происходящим в вычислительной системе, функционируя при этом в специальном режиме супервизора.

Каждый раз, когда приложению, выполняющемуся в защищенном режиме, требуется задействовать ресурсы оборудования, оно вынуждено формировать запрос и передавать его через механизм связи с ядром, которое в режиме

супервизора берет на себя непосредственное управление аппаратурой. Кроме того, нахождение программ в защищенном режиме накладывает существенные аппаратные ограничения на сами выполняемые машинные команды. В защищенном режиме программы могут иметь доступ только к более ограниченному набору инструкций процессора по сравнению с тем полным спектром команд, который доступен ядру в режиме супервизора. Подобное сокращение перечня разрешенных операций гарантирует, что программы в защищенном режиме не смогут выполнить потенциально опасные инструкции процессора, способные нарушить целостность работы оборудования или обойти механизмы связи с ядром.

Для обозначения и структурного описания этой иерархии уровней привилегий в технической литературе и документации используется дополнительная терминология, согласно которой в некоторых процессорах данные режимы работы также называются кольцевыми. Использование понятия кольцевых режимов в некоторых процессорах наглядно подчеркивает многоуровневую организацию защиты, где ядро, управляющее всем в режиме супервизора, находится на самом привилегированном уровне, а приложения, которые работают в защищенном режиме с доступом к более ограниченному набору инструкций процессора и используют оборудование только посредством связи с ядром, располагаются на менее привилегированном уровне.

Процессоры могут иметь и другие режимы, аналогичные

защищенному режиму, например, виртуальные режимы для эмуляции более старых типов процессоров, таких как 16-битные процессоры на 32-битном компьютере или 32-битные процессоры на 64-битном компьютере. Таким образом, операционная система может сохранять эксклюзивный контроль над такими вещами, как доступ к оборудованию и памяти. Архитектура современных центральных вычислительных устройств не ограничивается исключительно базовым разделением на защищенный режим и режим супервизора, поскольку процессоры могут иметь в своем составе и другие специализированные режимы работы.

Эти дополнительные режимы по своей внутренней организации и уровню накладываемых ограничений во многом являются аналогичными защищенному режиму. В качестве наиболее характерного и показательного примера подобных дополнительных состояний можно выделить специальные виртуальные режимы, которые аппаратно предназначены для обеспечения полноценной эмуляции более старых типов процессоров в рамках современной вычислительной среды.

Подобные виртуальные режимы позволяют без потери стабильности воспроизводить архитектуру и поведение предшествующих поколений электроники, например, обеспечивая корректную работу и эмуляцию старых 16-битных процессоров непосредственно на более современном 32-битном компьютере. Точно по такому же принципу аппарат-

ные виртуальные режимы, аналогичные защищенному режиму, применяются для воспроизведения среды 32-битных процессоров на высокопроизводительном 64-битном компьютере, сохраняя тем самым обратную совместимость программного обеспечения. Использование подобных виртуальных режимов в качестве безопасного аналога защищенного режима имеет принципиальное значение для поддержания целостности всей вычислительной среды.

Благодаря тому что программы старых поколений выполняются внутри таких изолированных и контролируемых виртуальных режимов для эмуляции более старых типов процессоров, операционная система может безоговорочно сохранять свой эксклюзивный контроль над критически важными ресурсами вычислительной машины. В частности, операционная система продолжает единолично и эксклюзивно контролировать такие фундаментальные вещи, как прямой доступ к физическому оборудованию и распределение оперативной памяти, не позволяя эмулируемым процессам 16-битных или 32-битных систем нарушать работу современного 32-битного или 64-битного компьютера.

2.5 Системные вызовы операционной системы

Системный вызов – это способ, которым программа запрашивает услугу у ядра операционной системы. Это может включать в себя услуги, связанные с оборудованием (например, доступ к жесткому диску), создание и выполнение новых процессов, а также взаимодействие с важными службами ядра, такими как планирование процессов.

Системные вызовы обеспечивают важный интерфейс между процессом и операционной системой. Системный вызов представляет собой фундаментальный программный механизм и строго регламентированный способ, посредством которого запущенная прикладная или фоновая программа официально запрашивает необходимую ей услугу непосредственно у ядра операционной системы.

В современной архитектуре вычислительных систем обычная компьютерная программа в силу действующих правил безопасности и разграничения уровней привилегий не может самостоятельно выполнять критически важные системные действия или напрямую распоряжаться физическими ресурсами машины.

Всякий раз, когда выполняющемуся процессу требуется выйти за пределы собственных изолированных вычислений и обратиться к внешним возможностям вычислитель-

ного комплекса, единственным законным и безопасным путем для него становится обращение к центральной управляющей части программной среды. Именно для реализации этого диалога существует системный вызов, который выступает тем самым базовым и незаменимым способом, которым программа запрашивает услугу у ядра операционной системы, передавая ему четкий сигнал о своих потребностях и ожидая корректного выполнения запрошенной операции.

Спектр задач, ради решения которых программа инициирует системный вызов и запрашивает услугу у ядра операционной системы, является чрезвычайно широким и охватывает ключевые аспекты функционирования всей вычислительной среды. Во-первых, подобное обращение за услугой к ядру операционной системы может напрямую включать в себя разнообразные услуги, неразрывно связанные с физическим оборудованием компьютера. Поскольку прямой контакт с аппаратными компонентами является исключительной прерогативой ядра операционной системы, любая программа, которой необходимо прочитать или записать данные, вынуждена использовать системный вызов для получения услуги, связанной с оборудованием.

В качестве наиболее характерного и наглядного примера подобных услуг, связанных с оборудованием, выступает организация доступа к жесткому диску, когда программа через системный вызов просит ядро операционной системы обратиться к накопителю для считывания или сохранения

информации, не имея права самостоятельно управлять контроллером жесткого диска. Во-вторых, помимо операций, затрагивающих аппаратную часть и доступ к жесткому диску, запрос услуги у ядра операционной системы посредством системного вызова может включать в себя жизненно важные операции по управлению жизненным циклом программ, такие как создание и последующее выполнение новых процессов.

Когда одной работающей программе требуется запустить другую программу или породить дополнительную вычислительную задачу, она не способна самостоятельно выделить для нее место в памяти и поставить в очередь на исполнение, поэтому она снова использует системный вызов, запрашивая у ядра операционной системы услугу по созданию и выполнению новых процессов. Ядро, получив такой запрос через системный вызов, берет на себя всю организационную работу по инициализации, выделению ресурсов и запуску новых процессов в строгом соответствии с правилами системы. В-третьих, функциональное наполнение системного вызова может также включать в себя непосредственное взаимодействие с другими важными внутренними службами ядра, среди которых особое место занимает такая критическая функция, как планирование процессов.

Когда программе необходимо скоординировать свою работу во времени, уступить вычислительные ресурсы или повлиять на порядок исполнения, она задействует системный

вызов для взаимодействия с важными службами ядра, такими как планирование процессов, позволяя ядру операционной системы корректно распределить время между всеми активными задачами. Таким образом, охватывая услуги, связанные с оборудованием, например доступ к жесткому диску, операции по созданию и выполнению новых процессов, а также прямое взаимодействие с важными службами ядра, такими как планирование процессов, механизм системных вызовов берет на себя роль универсального связующего звена. В конечном счете, именно системные вызовы обеспечивают исключительно важный, стандартизированный и надежный интерфейс между каждым отдельным работающим процессом и самой операционной системой.

Без наличия этого важного интерфейса, который формируют системные вызовы между процессом и операционной системой, ни один запущенный процесс не смог бы полноценно взаимодействовать с операционной системой, получать доступ к жесткому диску, инициировать создание и выполнение новых процессов или обращаться к таким важным службам ядра, как планирование процессов. Благодаря тому, что системные вызовы обеспечивают этот важный интерфейс между процессом и операционной системой, каждая программа получает четко определенный способ, которым она запрашивает любую необходимую услугу у ядра операционной системы, сохраняя при этом целостность, управляемость и безопасность всей вычислительной архитектуры.

2.6. Загрузчик операционной системы

Загрузчик – это компьютерная программа, которая загружает операционную систему или другое системное программное обеспечение для компьютера после завершения самотестирования при включении питания[1] ; это загрузчик самой операционной системы. В процессе жесткой перезагрузки он запускается после завершения самотестирования, затем загружает и запускает программное обеспечение. Загрузчик загружается в основную память из постоянной памяти, такой как жесткий диск или, в некоторых старых компьютерах, с носителя, такого как перфокарты, перфолента или магнитная лента. Затем загрузчик загружает и выполняет процессы, которые завершают загрузку. Как и процессы POST, код загрузчика поступает из *жестко заданного* и постоянного места; если это место по какой-либо причине слишком ограничено, основной загрузчик вызывает загрузчик второго этапа или вторичный загрузчик программ.

3. Работа программного обеспечения в концепции операционных систем

3.1. Стандартные потоки программ

Стандартные потоки – это предварительно подключенные каналы ввода-вывода между компьютерной программой и её окружением в начале выполнения. Три типа соединений ввода-вывода называются стандартным вводом (*stdin*), стандартным выводом (*stdout*) и стандартным выводом ошибок (*stderr*). Первоначально ввод-вывод осуществлялся через физически подключенную системную консоль (ввод через клавиатуру, вывод через монитор), но стандартные потоки абстрагируют этот процесс. При выполнении команды через интерактивную оболочку потоки обычно подключаются к текстовому терминалу, на котором работает оболочка, но их можно изменить с помощью перенаправления, например, через конвейер.

```
#include <iostream>

int main()
{
```

```
int n;

std::cin >> n;

float acc = 0.0f, x;

for (int ii = 0; ii < n; ++ii) {
std::cin >> x;
acc += x;
}

if (n <= 0) {
std::cerr << "Invalid number of elements" << std::endl;
} else {
const float avg = acc / n;
std::cout << avg << std::endl;
}

return 0;
}
```

3.2. Интерфейс прикладного программирования (API)

Программный интерфейс приложения (API) – это набор подпрограмм, протоколов и инструментов для создания программных приложений. API выражает программный компонент с точки зрения его операций, входных и выходных данных, а также базовых типов, определяя функциональные возможности, независимые от их соответствующих реализаций, что позволяет изменять определения и реализации без ущерба для интерфейса. Хороший API упрощает разработку программы, предоставляя все необходимые строительные блоки, которые затем программист собирает воедино. API может предоставляться операционной системой (например, Linux или Windows API), драйверами (например, Kinect SDK), языками программирования (например, Java, C++ или Swift) или сторонними библиотеками (например, Cocoa Touch).

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, купив полную легальную версию на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.