# Министерство образования Республики Беларусь

УПРАВЛЕНИЕ ОБРАЗОВАНИЯ МОГИЛЕВСКОГО ОБЛАСТНОГО ИСПОЛНИТЕЛЬНОГО КОМИТЕТА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Могилевский государственный политехнический колледж»

# Домашняя контрольная работа №1

По дисциплине: «**Системное программное обеспечение**»

Группа ПО-455

## Выполнил О. Е. Листопадова

Шифр 21

# 2022

**23 Дайте определение термину «Процесс»**

**Процесс** — это в выполняемая в данный момент программа. Выполнение процесса должно осуществляться последовательно. Процесс определяется как сущность, представляющая основную единицу работы, которая должна быть реализована в системе.

На любой ЭВМ всегда имеется процесс соответствующий операционной системе (ОС) этой ЭВМ, а также один или несколько процессов отвечающих пользовательским программам. На однопроцессорных ЭВМ в любой момент времени может выполнятся только один процесс. Любая ОС должна уметь производить запуск процессов, приостановку, их выполнение, завершение их выполнения и синхронизацию процессов между собой. Для каждого процесса ОС предоставляет собственное адресное пространство. Это адресное пространство начинается от нуля и продолжается непрерывно до предела соответствующего ЭВМ и ОС. С целью обеспечения переносимости адресное пространство всегда начинается с нуля. Ни один процесс кроме ОС не знает в какой именно части физической памяти и каким образом располагается его адресное пространство. Это прерогатива ОС, которая должна наиболее эффективным образом выполнять выполняющиеся процессы.

Операционная система контролирует следующую деятельность, связанную с процессами:

1. создание и удаление процессов
2. планирование процессов
3. синхронизация процессов
4. коммуникация процессов
5. разрешение тупиковых ситуаций

Каждому процессу ОС системой выделяются ресурсы: дисковое пространство, устройство ввода вывода, канал передачи информации и прочее. Каждый процесс имеет возможность создавать другие процессы и контролировать их выполнения. Каждому процессу в ОС отводятся определенные права. Эти права максимальны для самой ОС, имеют промежуточные значения для подсистем ОС (драйвера) и минимальные права соответствуют выполняющимся пользовательским программам. ОС для каждого из процесса хранит всю информацию об этих процессах в специальных таблицах. В этих таблицах обязательно описываются права процесса, полное состояние регистров процессора для данного процесса, объем ОП отводимый процессу и отображение этой памяти на реальную физическую память, а так же список всех ресурсов, отводимых данному процессу. При переключении с одного процесса на другой ОС пользуется этими учетными записями.

Порождение нового процесса это длительная процедура, так как ОС должна выполнить множество действий:

1. Выделить процессу необходимые ресурсы (адресное пространство, файлы, устройство и т.д.);
2. Произвести инициализацию этих ресурсов (загрузить выполняемую программу в ОП, инициализировать первое начальное значение регистров и стеков, открыть файлы и т.д.);
3. Занести всю необходимую информацию в специальную таблицу, описывающую процессу в системе;
4. Передать управление новому процессу.

Переключение между отдельными выполняющимися процессами так же длительная процедура. В этом случи ОС должна произвести дополнительные (обычно в таблице соответствующей выполняемому процессу) от регистров соответствующих выполняемому процессу, карту отображения в памяти процесса на реальную физическую память, сохранить состояние всех файлов и устройств используемых процессом. После этого ОС должна выбрать полное описание процесса на который она переключается из таблицы (инициализировать регистры, карту отображения памяти процесса на физическую память, состояние файлов и устройств). Переключение между процессами осуществляется в том числе по прерыванию таймера. Обычно системному программисту предоставляется возможность задания максимального времени выполнения одного процесса, после которого произойдет прерывание по таймеру и переключение на другой процесс. При задании маленького значения этого времени у системы будет мало времени на отклик на возникающие события, однако при этом значительная часть процессорного времени будет тратиться на переключение между процессами. При задании большого времени таймера накладные расходы, связанные с переключениями между процессами будут уменьшаться, однако будет ухудшаться отклик системы на возникающие события. При завершении процесса происходит закрытие всех файлов, освобождение всех ресурсов, занятых всех ресурсов и вычеркивании его из таблицы процессов.

Планирование процессов необходимо для организации более производительной работы многозадачной, многопользовательской ОС. В однозадачный однопользовательских ОС, система не ведет никакого планирования запуска на выполнения отдельных процессов. Все задачи планирования выполняет пользователь, работающий за однозадачной однопользовательской ОС. Однако для многозадачных многопользовательских ОС (UNIX) есть необходимость в таком планировании, так как в очереди на выполнение обычно стоит большое количество различных процессов. Планирование проявляется так же в выделении различных приоритетов, объема ОП, количество выделяемых ресурсов и процессорного времени, предоставляемого отдельным процессам.

**50 Охарактеризуйте классы дисциплин обслуживания процессов**

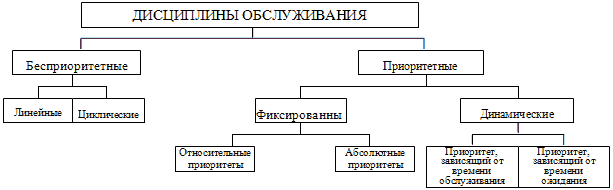
Различают приоритетные и бесприоритетные дисциплины обслуживания.

Бесприоритетные ДО - выбор из очереди производится без учета относительной важности задач и времени их обслуживания. Приоритетное обслуживание - отдельным задачам предоставляется преимущественное право перейти в состояние ВЫПОЛНЕНИЯ.

Приоритеты, в свою очередь, могут быть фиксированными или динамическими:

Фиксированные приоритеты - являются величиной постоянной на всем жизненном цикле процесса. Динамические приоритеты - изменяются в зависимости от некоторых условий в соответствии с определенными правилами.

Фиксированные приоритеты реализуются проще; для реализации динамических приоритетов необходимы дополнительные затраты, но их использование предполагает более справедливое распределение процессорного времени между процессами. Классификация дисциплин обслуживания представлена на рис. 1

  
Рис.1. Классификация дисциплин обслуживания

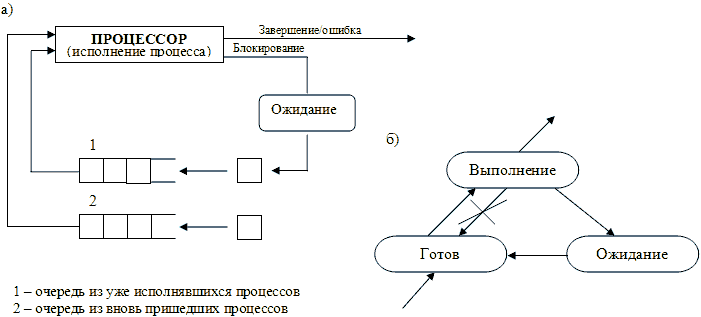
**Описание дисциплин обслуживания**

Бесприоритетная линейная дисциплина обслуживания

Особенности организации: Реализация стратегии, при которой задания заканчиваются в порядке поступления. Две очереди готовых процессов: одна - для уже выполнявшихся процессов, другая - для вновь пришедших.

Схема линейной дисциплины обслуживания и граф состояний процесса в системе с линейной дисциплиной обслуживания представлены на рис 2.

Перераспределения процессорного времени не происходит, т. е. операции принудительного прекращения процесса в этой схеме

  
Рис.2. а) Схема линейной дисциплины обслуживания;

б) Граф состояний процесса в системе с линейной дисциплиной обслуживания

Перераспределения процессорного времени не происходит, т. е. операции принудительного прекращения процесса в этой схеме нет. Процесс, начавший выполнение вытеснен быть не может. Выполнившиеся процессы ставятся в конец очереди 1 (уже выполнявшихся). Всегда выбирается первый процесс из очереди. Сначала выбираются процессы из очереди 1 (уже выполнявшихся); процессы из очереди 2 (вновь поступивших) выбираются, если очередь 1 (уже выполнявшихся) пуста.

Достоинства:

Исключительная простота реализации Малый расход системных ресурсов на организацию

Недостатки:

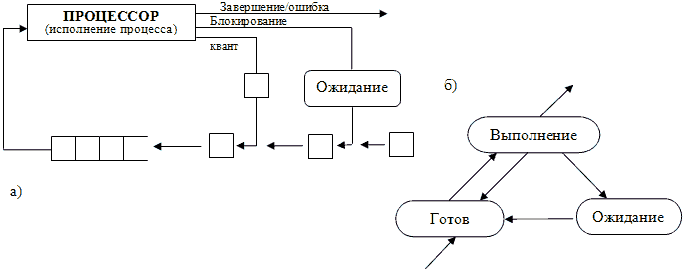
При увеличении нагрузки растет время ожидания обслуживания, при этом короткие задания ждут столько же, сколько и длинные. Следовательно, имеет место дискриминация процессов, постоянно присутствует вероятность откладывания обслуживания.

**Бесприоритетная циклическая (карусельная) дисциплина обслуживания**

Особенности организации: Основана на квантовании. Процесс может быть вытеснен по истечению кванта, если он к этому времени не закончился. Он ставится в конец очереди наряду с поступившими процессами. Очередь готовых процессов одна - для уже выполнявшихся и для вновь поступивших. Из очереди всегда выбирается первый процесс, ставится в конец очереди. Смена выполняющегося процесса может произойти в следующих случаях:

процесс закончил свое выполнение или произошла ошибка процесс перешел в состояние ОЖИДАНИЕ истек квант времени, отведенный процессу

Схема циклической дисциплины обслуживания и граф состояний процесса в системе с циклической дисциплиной обслуживания представлены на рис 3.

  
Рис.3. а) Схема циклической дисциплины обслуживания;  
б) граф состояний процесса в системе с циклической дисциплиной обслуживания.

Достоинства:

При относительной простоте реализации исключено бесконечное откладывание обслуживания

Недостатки:

Увеличение доли накладных расходов при уменьшении кванта и ухудшение реакции системы при увеличении кванта.

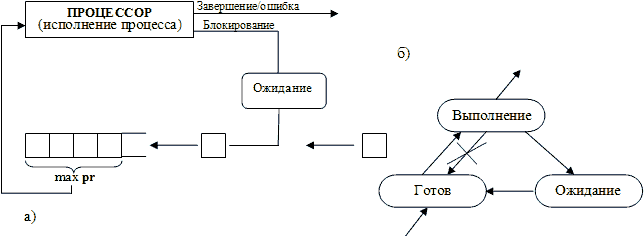
**Дисциплина обслуживания с относительными приоритетами**

Особенности организации: Дисциплина обслуживания, основанная на приоритетах. Процесс не может быть вытеснен другими заданиями. Очередь готовых процессов одна. Процесс всегда ставится в конец очереди. На исполнение из очереди выбирается процесс с наибольшим приоритетом.

Смена выполняющегося задания происходит в следующих случаях:

процесс завершен или произошла ошибка; процесс перешел в состояние ожидания.

Схема дисциплины обслуживания с относительными приоритетами и граф состояний процесса в системе с дисциплиной обслуживания с относительными приоритетами представлены на рис 4.

  
Рис.4. а) Схема дисциплины обслуживания с относительными приоритетами;  
б) Граф состояний процесса в системе с дисциплиной обслуживания с относительными приоритетами.

Достоинства:

учитывается приоритетность задач

Недостатки:

вероятность бесконечного откладывания обслуживания потеря контроля над системой (контроль теряется в тот момент, когда управление передается процессу).

**Дисциплина обслуживания с абсолютными приоритетами**

Особенности организации: Дисциплина обслуживания, основанная на приоритетах. Процесс может быть вытеснен процессом с большим приоритетом. Процессы ставятся в конец единственной очереди. Из очереди выбирается процесс с максимальным приоритетом

Смена выполняющегося задания происходит в следующих случаях:

процесс завершен или произошла ошибка; процесс перешел в состояние ожидания; в очереди появился процесс с большим приоритетом

Достоинства:

Учитывает приоритетность задач. Не теряется контроль над системой (в любой момент можно запустить системную задачу с наивысшим приоритетом).

Недостатки:

Вероятность бесконечного откладывания обслуживания низкоприоритетных задач. Очень сложная реализация, поскольку сложно определить момент для пересчета приоритетов

Схема дисциплины обслуживания с абсолютными приоритетами и граф состояний процесса в системе с соответствующей дисциплиной обслуживания представлены на рис 5.

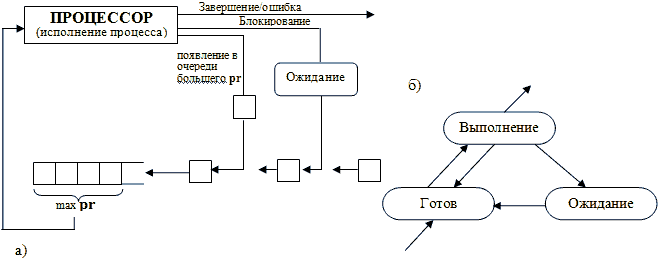


Рис.5. а) Схема дисциплины обслуживания с абсолютными приоритетами;  
б) Граф состояний процесса в системе с дисциплиной обслуживания с абсолютными приоритетами.

**Дисциплина обслуживания с приоритетом, зависящем от времени обслуживания**

Особенности организации: Дисциплина обслуживания, основанная на абсолютных приоритетах. Во время выполнения процесса его приоритет уменьшается с каждым тиком. Если приоритет процесса становится меньше приоритета процесса стоящего в очереди готовых, процесс будет вытеснен с выполнения. Это позволяет уменьшить дискриминацию процессов, возникающую при использовании дисциплин обслуживания с абсолютными приоритетами.

Смена выполняющегося задания происходит в следующих случаях:

процесс завершен или произошла ошибка; процесс перешел в состояние ожидания;

приоритет задания становится меньше, чем у ожидающего в очереди готовых задания с наибольшим приоритетом в очереди появился процесс с большим приоритетом

Схема дисциплины обслуживания с приоритетами, зависящими от времени выполнения, и граф состояний процесса в системе с соответствующей дисциплиной обслуживания представлены на рис 6.

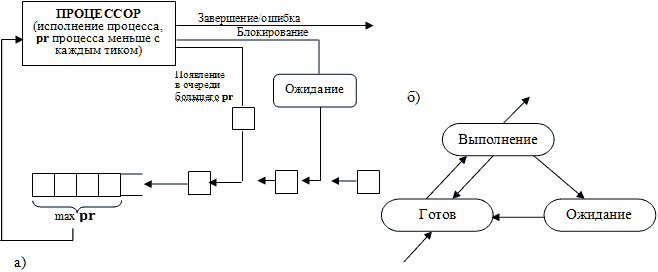


Рис.6. а) Схема дисциплины обслуживания с приоритетами, зависящими от времени выполнения;

б) Граф состояний процесса в системе с дисциплиной обслуживания с приоритетами, зависящими от времени выполнения

Достоинства:

Учитывает приоритетность задач Уменьшает возможность недобросовестного использования механизмов приоритетов

Недостатки:

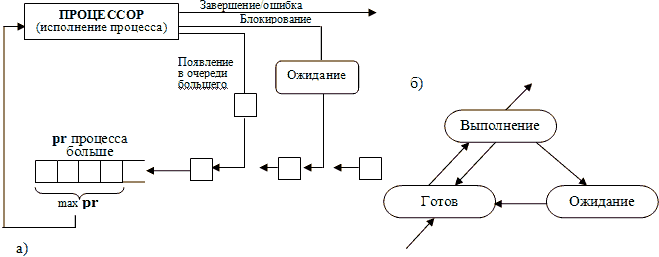
- Возможность бесконечного откладывания низкоприоритетных процессов - Сложная организация, так как необходим пересчет приоритетов.

**Дисциплина обслуживания с приоритетом, зависящем от времени ожидания в очереди готовых процессов**

Особенности организации: Дисциплина обслуживания, основанная на абсолютных приоритетах. По мере ожидания в очереди готовых, приоритет процесса увеличивается с каждым тиком. Если приоритет выполняющегося процесса становится меньше приоритета процесса стоящего в очереди готовых (независимо от того, новый это процесс или давно стоящий), процесс будет вытеснен с выполнения. Это позволяет исключить дискриминацию процессов, возникающую при использовании дисциплин обслуживания с абсолютными приоритетами и дисциплин обслуживания с приоритетами, зависящими от времени выполнения. Смена выполняющегося задания происходит в следующих случаях:

процесс завершен или произошла ошибка; процесс перешел в состояние ожидания; приоритет выполняющегося задания становится меньше, чем у ожидающего в очереди готовых задания с наибольшим приоритетом; в очереди появился процесс с большим приоритетом.

Схема дисциплины обслуживания с приоритетами, зависящими от времени ожидания, и граф состояний процесса в системе с соответствующей дисциплиной обслуживания представлены на рис 7.

Рис.7. а) Схема дисциплины обслуживания с приоритетами, зависящими от времени ожидания;

б) Граф состояний процесса в системе с дисциплиной обслуживания с приоритетами, зависящими от времени ожидания

**88 Перечислите основные функции драйвера**

Драйвер — компьютерное программное обеспечение, с помощью которого другое программное обеспечение (операционная система) получает доступ к аппаратному обеспечению некоторого устройства. Обычно с операционными системами поставляются драйверы для ключевых компонентов аппаратного обеспечения, без которых система не сможет работать. Однако для некоторых устройств (таких, как видеокарта или принтер) могут потребоваться специальные драйверы, обычно предоставляемые производителем устройства.

Разнообразный набор драйверов для широкого круга популярных периферийных устройств – непременное условие популярности ОС у пользователей.

Для разработки драйверов производителями внешних устройств необходимо наличие четкого, удобного, открытого и хорошо документированного интерфейса между драйверами и другими компонентами ОС. Драйвер взаимодействует, с одной стороны, с модулями ядра ОС (модулями подсистемы ввода-вывода, модулями системных вызовов, модулями подсистем управления процессами и памятью), а с другой стороны – с контроллерами внешних устройств. Поэтому существует два вида интерфейсов: интерфейс «драйвер-ядро» (Driver Kernel Interface, DKI) и интерфейс «драйвер-устройство» (Driver Device Interface).

Интерфейс «драйвер-ядро» должен быть стандартизован в любом случае. Подсистема ввода-вывода может поддерживать несколько различных интерфейсов DKI/DDI, предоставляя специфический интерфейс для устройств определенного класса. К наиболее общим классам относятся блочные устройства, например, диски, и символьные устройства, такие как клавиатура и принтеры. Может существовать класс сетевых адаптеров и др. В большинстве современных ОС определен стандартный интерфейс, который должен поддерживать все блочные драйверы, и второй стандартный интерфейс, поддерживаемый всеми символьными адаптерами. Эти интерфейсы включают наборы процедур, которые могут вызываться остальной операционной системой для обращения к драйверу. К этим процедурам относятся, например, процедуры чтения блока или записи символьной строки.

Кроме того, подсистема ввода-вывода поддерживает большое количество системных функций, которые драйвер может вызывать для выполнения некоторых типовых действий. Например, это операции обмена с регистрами контроллера, ведения буферов промежуточного хранения данных ввода-вывода, взаимодействия с DMA-контроллером и контроллером прерываний и др.

У драйверов устройств есть множество функций:

1. Обработка запросов записи-чтения от программного обеспечения управления устройствами. Постановка запросов в очередь.

2. Проверка входных параметров запросов и обработка ошибок.

3. Инициализация устройства и проверка статуса устройства.

4. Управление энергопотреблением устройства.

5. Регистрация событий в устройстве.

6. Выдача команд устройству и ожидание их выполнения, возможно, в блокированном состоянии, до поступления прерывания от устройства.

7. Проверка правильности завершения операции.

8. Передача запрошенных данных и статуса завершенной операции.

9. Обработка нового запроса при незавершенном предыдущем запросе (для реентерабельных драйверов).

Наиболее очевидная функция состоит в обработке абстрактных запросов чтения и записи независимого от устройств программного обеспечения, расположенного над ними. Но, кроме этого, они должны выполнять еще несколько функций. Например, драйвер должен при необходимости инициализировать устройство. Ему может понадобиться управлять энергопотреблением устройства и регистрацией событий.

Многие драйверы обладают сходной общей структурой. Типичный драйвер начинает работу с проверки входных параметров. Если они не удовлетворяют определенным критериям, драйвер возвращает ошибку. В противном случае драйвер преобразует абстрактные термины в конкретные. Например, дисковый драйвер преобразует линейный номер кластера в номер головки, дорожки и сектора.

Затем драйвер может проверить, не используется ли это устройство в данный момент. Если устройство занято, запрос может быть поставлен в очередь. Если устройство свободно, проверяется статус устройства, чтобы понять, может ли запрос быть обслужен прямо сейчас. Может оказаться необходимым включить устройство или запустить двигатель, прежде чем начнется перенос данных. Как только устройство включено и готово, начинается собственно управление устройством.

Управление устройством подразумевает выдачу ему серии команд. Именно в драйвере определяется последовательность команд в зависимости от того, что должно быть сделано. Определившись с командой, драйвер начинает записывать их в регистры контроллера устройства. После записи каждой команды в контроллер, возможно, будет нужно проверить, принял ли контроллер команду и готов ли принять следующую. Такая последовательность действий продолжается до тех пор, пока контроллеру не будут переданы все команды. Некоторые контроллеры способны принимать связные списки команд, находящихся в памяти. Они сами считывают и выполняют их без дальнейшей помощи ОС.

После того как драйвер передал все команды контроллеру, ситуация может развиваться по двум сценариям. Во многих случаях драйвер устройства должен ждать, пока контроллер не выполнит для него определенную работу, поэтому он блокируется до тех пор, пока прерывание от устройства не разблокирует его. В других случаях операция завершается без задержек и драйверу не нужно блокироваться. В любом случае по завершении выполнения операции драйвер должен проверить, завершилась ли операция без ошибок. Если все в порядке, драйверу, возможно, придется передать данные (например, только что прочитанный блок) независимому от устройств программному обеспечению. Наконец, драйвер возвращает информацию о состоянии для информирования вызывающей программы о статусе завершения операции. Если в очереди находились другие запросы, один из них теперь может быть выбран и запущен. В противном случае драйвер блокируется в ожидании следующего запроса.

Для поддержки процесса разработки драйверов операционной системы выпускается так называемый пакет DDK (Driver Development Kit), представляющий собой набор инструментальных средств-библиотек, компиляторов и отладчиков.

Так как набор потенциально поддерживаемых данных ОС периферийных устройств всегда шире набора устройств, которыми ОС должна управлять при установке на конкретной машине, то ценным свойством ОС является возможность динамически загружать в оперативную память требуемый драйвер (без остановки ОС) и выгружать его, если надобность в драйвере отпала. Такое свойство ОС может существенно сэкономить системную область памяти.

Альтернативой динамической загрузке драйверов при изменении текущей конфигурации внешних устройств компьютера является повторная компиляция кода ядра с требуемым набором драйверов, что создает между всеми компонентами ядра статические связи вместо динамических. Например, таким образом решалась данная проблема в ранних версиях ОС UNIX. При статистических вызовах между ядром и драйверами структура ОС упрощается, но этот подход требует наличия исходных кодов модулей ОС, доступность которых скорее является исключением (для некоммерческих версий UNIX). Кроме того, в этом варианте работающую версию ОС надо остановить и заменить новой, что не всегда допустимо в некоторых применениях.

Поэтому поддержка динамической загрузки драйверов является практически обязательным требованием для современных универсальных ОС.

Первоначально термин «драйвер» применялся в достаточно узком смысле – под драйвером понимается программный модуль, который:

- входит в состав ядра ОС, работая в привилегированном режиме;

- непосредственно управляет внешним устройством, взаимодействуя с его контроллером с помощью команд ввода-вывода компьютера;

- обрабатывает прерывания от контроллера устройства;

- предоставляет прикладному программисту удобный логический интерфейс работы с устройством, экранируя от него низкоуровневые детали управления устройством и организации его данных;

- взаимодействует с другими модулями ядра ОС с помощью строго оговоренного интерфейса, описывающего формат передаваемых данных, структуру буферов, способы включения драйвера в состав ОС, способы вызова драйвера, набор общих процедур подсистемы ввода-вывода, которыми драйвер может пользоваться и т.п.

Согласно этому определению драйвер вместе с контроллером устройства и прикладной программой воплощали идею многослойного подхода к организации программного обеспечения. Контроллер представлял низкий слой управления устройством, выполняющий операции в терминах блоков и агрегатов устройства (например, передвижение головки дисковода, побитную передачу байта по двухпроводному кабелю). Драйвер выполнял более сложные операции, преобразуя данные, адресуемые в терминах номеров цилиндров, головок и секторов диска, в линейную последовательность блоков. В результате прикладная программа работала с данными, преобразованными в достаточно понятную форму, – файлами, таблицами баз данных, текстовыми окнами на мониторе и т.п., не вдаваясь в детали представления этих данных в устройствах ввода-вывода.

В описанной схеме драйверы не делились на слои. Постепенно, по мере развития операционных систем и усложнения структуры подсистемы ввода-вывода, наряду с традиционными драйверами в ОС появились так называемые высокоуровневые драйверы, которые располагаются в общей модели подсистемы ввода-вывода над традиционными драйверами. Появление таких драйверов можно считать развитием идеи многоуровневой организации подсистемы ввода-вывода, когда ее функции декомпозируются между несколькими модулями в соседних слоях иерархии (таких примеров много, например семиуровневая модель сетевых протоколов).

Традиционные драйверы, которые стали называть аппаратными, низкоуровневыми или драйверами устройств, освобождаются от высокоуровневых функций и занимаются только низкоуровневыми операциями. Эти низкоуровневые операции составляют фундамент, на котором можно построить тот или иной набор операций в драйверах более высоких уровней.

При таком подходе повышается гибкость и расширяемость функции по управлению устройством. Например, если различным приложениям необходимо работать с различными логическими модулями одного и того же физического устройства, то для этого в системе достаточно установить несколько драйверов на одном уровне, работающих над одним аппаратным драйвером. Несколько драйверов, управляющих одним устройством, но на разных уровнях, можно рассматривать как один многоуровневый драйвер.

На практике используют от двух до пяти уровней драйверов, поскольку с увеличением числа уровней снижается скорость выполнения операций ввода-вывода.

Высокоуровневые драйверы оформляются по тем же правилам и придерживаются тех же внутренних интерфейсов, что и аппаратные драйверы. Как правило, высокоуровневые драйверы не вызываются по прерываниям, так как взаимодействуют с устройством через посредничество аппаратных драйверов.

В модулях подсистемы ввода-вывода, кроме драйверов, могут присутствовать и другие модули, например, дисковый кэш. Достаточно специфичные функции кэша делают нецелесообразным оформление его в виде драйвера, взаимодействующего с другими модулями ОС только с помощью услуг менеджера ввода-вывода. Другим примером модуля, который чаще всего не оформляется в виде драйвера, является диспетчер окон графического интерфейса. Иногда этот модуль вообще выносится из ядра ОС и реализуется в виде пользовательского интерфейса. Таким образом, был реализован диспетчер окон в Windows NT 3.5 и 3.51, но этот микроядерный подход заметно замедляет графические операции, поэтому в Windows 4.0 диспетчер окон и высокоуровневые графические драйверы, а также графическая библиотека GDI были перенесены в пространство ядра.

Аппаратные драйверы после запуска операции ввода-вывода должны своевременно реагировать на завершение контроллером заданного действия путем взаимодействия с системой прерывания. Драйверы более высоких уровней вызываются не по прерываниям, а по инициативе аппаратных драйверов или драйверов вышележащего уровня. Не все процедуры аппаратного драйвера нужно вызывать по прерываниям, поэтому драйвер обычно имеет определенную структуру, в которой выделяется секция обработки прерываний (Interrupt Service Routine, ISR), которая и вызывается от соответствующего устройства диспетчером прерываний.

В унификацию драйверов большой вклад внесла ОС UNIX, в которой все драйверы были разделены на два класса: блок-ориентированные (Block-oriented) и байт-ориентированные (Character-oriented) драйверы. Это более общее деление, чем деление на вертикальные подсистемы. Например, драйверы графических устройств и сетевых устройств относятся к классу байт-ориентированных.

Блок-ориентированные драйверы управляют устройствами прямого доступа, которые хранят информацию в блоках фиксированного размера, каждый из которых имеет свой адрес. Адресуемость блоков приводит к тому, что для дисков, являющихся устройствами прямого доступа, появляется возможность кэширования данных в оперативной памяти. Это обстоятельство значительно влияет на общую организацию ввода-вывода для блок-ориентированных драйверов.

Устройства, с которыми работают байт-ориентированные драйверы, не адресуют данные и не позволяют производить операции поиска данных, они генерируют или потребляют последовательность байта (терминалы, принтеры, сетевые адаптеры и т.п.).

Однако не все устройства, управляемые подсистемой ввода-вывода, можно разделить на блок и байт-ориентированные. Для таких устройств (например, таймер) нужен специфический драйвер.

В свое время ОС UNIX сделала очень важный шаг по унификации операций и структуризации программного обеспечения ввода-вывода. В ОС UNIX все устройства рассматриваются как виртуальные (специальные) файлы, что дает возможность использовать общий набор базовых операций ввода-вывода для любых устройств независимо от их специфики. Подобная идея реализована позже в MS-DOS, где последовательные устройства – монитор, принтер и клавиатура – считаются файлами со специальными именами: con, prn, con.

**112 Опишите подсистему учетных записей**

Между включением питания компьютера и моментом, когда система готова к работе с пользователем, происходит процедура **загрузки системы**. В процессе загрузки будет запущена основная управляющая программа (ядро), определено и инициализировано имеющееся оборудование, активизированы сетевые соединения, запущены системные службы. В Linux во время загрузки на экран выводятся диагностические сообщения о происходящих событиях, и если всё в порядке и не возникло никаких ошибок, загрузка завершится выводом на экран приглашения «login:». Оно может быть оформлено по-разному, в зависимости от настройки системы оно может отображаться в красиво оформленном окне или в виде простой текстовой строки вверху экрана. Это приглашение к **регистрации в системе**: система ожидает, что в ответ на это приглашение будет введено **входное имя пользователя**, который начинает работу. Естественно, имеет смысл вводить такое имя, которое уже известно системе, чтобы она могла «узнать», с кем предстоит работать, выполнять команды «незнакомого» Linux откажется.

## Многопользовательская модель разграничения доступа

Процедура регистрации в системе обязательна для Linux, работать в системе, не зарегистрировавшись под тем или иным именем пользователя, просто невозможно. Для каждого пользователя определена сфера его полномочий в системе: программы, которые он может запускать, файлы, которые он имеет право просматривать, изменять, удалять. При попытке сделать что-то, выходящее за рамки полномочий, пользователь получит сообщение об ошибке. Такая строгость может показаться необязательной, если пользователи компьютера доверяют друг другу, и особенно если у компьютера только один пользователь. Такая ситуация очень распространена на сегодняшний день, когда слово «компьютер» означает в первую очередь «персональный компьютер».

Вместо формального «зарегистрироваться в системе» обычно используют выражение «войти в систему». Операционная система представляется чем-то вроде замкнутого помещения, внутри которого можно оказаться, только успешно проникнув через «дверь» — пройдя процедуру регистрации.

Однако **персональный компьютер** — довольно-таки позднее явление в мире вычислительной техники, получившее широкое распространение только в последние два десятилетия. Несколько раньше «компьютер» ассоциировался с огромным и дорогостоящим (занимавшем целые залы) вычислительным центром, предназначенным в первую очередь для решения разного рода научных задач. Машинное время такого центра стоит очень недёшево, и при этом его возможности необходимы одновременно многим сотрудникам, которые могут ничего не знать о работе друг друга. Требуется следить за тем, чтобы не произошло случайного вмешательства пользователей в чужую работу и повреждения чужих данных (файлов), выделять каждому машинное время (по возможности избежав простаивания), пространство на диске и при этом не допустить узурпирования всех ресурсов одним пользователем и его задачей, а равномерно делить ресурсы между всеми. Для такой системы принципиально важно знать, кому принадлежат задачи и файлы, поэтому и возникла необходимость выдавать доступ к ресурсам системы только после того, как пользователь зарегистрируется в системе под тем или иным именем. Такая модель была реализована в **многопользовательской операционной системе** UNIX. Именно от неё Linux — также многопользовательская система — унаследовал принципы работы с пользователями. Но это не просто дань традиции или стремление к универсальности: многопользовательская модель позволяет решить ряд задач, весьма актуальных и для современных **персональных компьютеров**, и для серверов, работающих в локальных и глобальных сетях, и вообще в любых системах, одновременно выполняющих разные задачи, отвечают за которые разные люди. Компьютер — это всего лишь инструмент для решения разного рода прикладных задач: от набора и распечатывания текста до вычислений. Сложность состоит в том, что для изменения этого инструмента и для работы с его помощью используются одни и те же операции: изменение файлов, выполнение программ. Получается, что, если не соблюдать осторожности, побочным результатом работы может стать выход из строя самой системы. Поэтому первоочередная задача для систем любого масштаба — разделять повседневную работу и изменение самой системы. В многопользовательской модели эта задача решается очень просто: разделяются **обычные пользователи** и **администратор**. В полномочия обычного пользователя входит все необходимое для выполнения прикладных задач, попросту говоря, для работы, однако ему запрещено выполнение действий, изменяющих саму систему. Таким образом можно избежать повреждения системы в результате ошибки пользователя (нажал не ту кнопку) или ошибки в программе, или даже по злому умыслу (например, вредительской программой-вирусом). Полномочия администратора обычно не ограничены. Для персонального компьютера, с которым работают несколько человек, довольно важно обеспечить каждому независимую рабочую среду. Это снижает вероятность случайного повреждения чужих данных, а также позволяет каждому пользователю настроить внешний вид рабочей среды по своему вкусу и, например, сохранить расположение открытых окон между сеансами работы. Эта задача очевидным образом решается в многопользовательской модели: организуется **домашний каталог**, где хранятся данные пользователя, настройки внешнего вида и поведения его системы и т. п., доступ остальных пользователей к этому каталогу ограничивается. Если компьютер подключён к глобальной или локальной сети, то вполне вероятно, что какую-то часть хранящихся на нем ресурсов имеет смысл сделать публичной и доступной по сети. Напротив, часть данных, скорее всего, делать публичными не следует (например, личную переписку). Ограничив публичный доступ пользователей к персональным данным друг друга, мы решим и эту задачу. Именно благодаря гибкости многопользовательской модели разграничения доступа она используется сегодня не только на серверах, но и на домашних персональных компьютерах. В самом простом варианте — для персонального компьютера, на котором работает только один человек — эта модель сводится к двум пользователям: обычному пользователю для повседневной работы и администратору — для настройки, обновления, дополнения системы и исправления неполадок. Но даже в таком сокращённом варианте это даёт целый ряд названных выше преимуществ.

## Учётные записи

Конечно, система может быть «знакома» с человеком только в переносном смысле: в ней должна храниться запись о пользователе с таким именем и о связанной с ним системной информации — **учётная запись**. Английский эквивалент термина **учётная запись** — **account**, «счёт». Именно с учётными записями, а не с самими пользователями, и работает система. В действительности, соотношение учётных записей и пользователей-людей в Linux обычно не является однозначным: несколько человек могут использовать одну учётную запись — система не может их различить. И в то же время в Linux имеются учётные записи для **системных пользователей**, от имени которых работают некоторые программы и которые не предназначены для работы людей.

Учётная запись это объект системы, при помощи которого Linux ведёт учёт работы пользователя в системе. Учётная запись содержит данные о пользователе, необходимые для регистрации в системе и дальнейшей работы с ней.

Учётные записи могут быть созданы во время установки системы или после установки. Подробно процедура создания учётных записей (добавления пользователей) описана в лекции [Конфигурационные файлы](https://linuxcookbook.ru/books/textbooks/linux_intro/Etc.html). Главное для человека в учётной записи — её название, **входное имя пользователя**. Именно о нём спрашивает система, выводя приглашение «login:». Помимо входного имени в учётной записи содержатся некоторые сведения о пользователе, необходимые системе для работы с ним. Ниже приведён список этих сведений.

Входное имя: название учётной записи пользователя, которое нужно вводить при регистрации пользователя в системе.

### Идентификатор пользователя

Linux связывает **входное имя** c **идентификатором пользователя** в системе — **UID** (User ID). **UID** — это положительное целое число, по которому система и отслеживает пользователей.

Это может оказаться важным, например, в такой ситуации: учётную запись пользователя с именем test удалили из системы, а потом добавили снова. Однако с точки зрения системы это уже другой пользователь, потому что у него другой UID.

Обычно это число выбирается автоматически при регистрации учётной записи, однако оно не может быть совершенно произвольным. В Linux есть некоторые соглашения относительно того, каким типам пользователей могут быть выданы идентификаторы из того или иного диапазона. В частности, **UID** от «0» до «100» зарезервированы для **псевдопользователей**.

Обычно Linux выдаёт нормальным пользователям UID, начиная с «500» или «1000».

Идентификатор пользователя это уникальное число, однозначно идентифицирующее **учётную запись** пользователя в Linux. Таким числом снабжены все **процессы** Linux и все объекты **файловой системы**. Используется для персонального учёта действий пользователя и определения **прав доступа** к другим объектам системы

### Идентификатор группы

Кроме идентификационного номера пользователя с учётной записью связан **идентификатор группы**. **Группы пользователей** применяются для организации доступа нескольких пользователей к некоторым ресурсам. У группы, так же, как и у пользователя, есть имя и идентификационный номер — **GID** (Group ID). В Linux пользователь должен принадлежать как минимум к одной группе — **группе по умолчанию**. При создании учётной записи пользователя обычно создаётся и группа, имя которой совпадает с **входным именем**, именно эта группа будет использоваться как группа по умолчанию для этого пользователя. Пользователь может входить более чем в одну группу, но в учётной записи указывается только номер группы по умолчанию.

Как правило, численное значение GID в этом случае совпадает со значением UID.

### Полное имя

Помимо **входного имени** в учётной записи содержится и **полное имя** (имя и фамилия) использующего данную учётную запись человека. Конечно, пользователь может указать что угодно в качестве своего имени и фамилии. Полное имя необходимо не столько системе, сколько людям — чтобы иметь возможность определить, кому принадлежит учётная запись.

### Домашний каталог

Файлы всех пользователей в Linux хранятся раздельно, у каждого пользователя есть собственный **домашний каталог**, в котором он может хранить свои данные. Доступ других пользователей к домашнему каталогу пользователя может быть ограничен. Информация о домашнем каталоге обязательно должна присутствовать в учётной записи, потому что именно с него начинает работу пользователь, зарегистрировавшийся в системе.

### Командная оболочка

Каждому пользователю нужно предоставить способ взаимодействовать с системой: передавать ей команды и получать её ответы. Для этой цели служит специальная программа — **командная оболочка** (или **интерпретатор командной строки**), она должна быть запущена для каждого пользователя, зарегистрировавшегося в системе. Поскольку в Linux доступно несколько разных командных оболочек, в учётной записи указано, какую из командных оболочек нужно запустить для данного пользователя. Если специально не указывать командную оболочку при создании учётной записи, она будет назначена по умолчанию, вероятнее всего это будет bash.

интерпретатор командной строки

Программа, используемая в Linux для огранизации диалога человека и системы. Командный интерпретатор имеет три основных ипостаси: (1) редактор и анализатор команд в **командной строке**, (2) высокоуровневый системно-ориентированный язык программирования, (3) средство организации взаимодействия команд друг с другом и с системой.

## Понятие «администратор»

В Linux есть ровно один пользователь, полномочия которого в системе принципиально отличаются от полномочий остальных пользователей — это пользователь с идентификатором «0». Обычно учётная запись пользователя с UID=0 называется root (англ., «корень»). Пользователь root — это «администратор» системы Linux, учётная запись для root обязательно присутствует в любой системе Linux, даже если в ней нет никаких других учётных записей. Пользователю с таким UID разрешено выполнять любые действия в системе, а значит, любая ошибка или неправильное действие может повредить систему, уничтожить данные и привести к другим печальным последствиям. Поэтому категорически не рекомендуется регистрироваться в системе под именем root для повседневной работы. Работать в root следует только тогда, когда это действительно необходимо: при настройке и обновлении системы, восстановлении после сбоев. Именно root обладает достаточными полномочиями для создания новых учётных записей.

Когда система выводит на экран приглашение командной строки после того, как правильно введены имя пользователя и пароль, это означает, что произошла **идентификация пользователя** (authentication, «проверка подлинности»). Пароль может показаться излишней сложностью, но у системы нет другого способа удостовериться, что за монитором находится именно тот человек, который имеет право на использование данной учётной записи. Конечно, процедура идентификации имеет очевидное значение для систем, к которым имеют непосредственный или сетевой доступ многие не связанные друг с другом пользователи. Процедура идентификации даёт уверенность, что к такой системе не получит доступ случайный человек, не имеющий права использовать её ресурсы и хранящуюся там информацию. Одновременно она даёт определённую гарантию безопасности от злонамеренного вмешательства: даже если навредить попытается пользователь, имеющий учётную запись, его действия будут зарегистрированы в системе (поскольку системе всегда известно, от имени какой учётной записи выполняются те или иные действия), и злоумышленника можно будет найти и остановить.

**Список используемых источников**

1. Артамонова, Н.В. Операционные системы для организации производства в промышленности/Н.В. Артамонова – СПб: ГУАП, 2012.
2. Гордеев, А.В. Системное программное обеспечение / А.В.Гордеев, А.Ю. Молчанов– СПб: Питер, 2003.
3. Дроздов, С.Н. Операционные системы: учебное пособие / С.Н.Дроздов. – РнД: Феникс, 2016.
4. Иртегов, Д. Введение в операционные системы / Д. Иртегов – СПб: BHV , 2012.
5. Коньков, К.А. Устройство и функционирование ОС Windows / К.А. Коньков – М: Бином, 2012.
6. Партыка, Т.Л. Операционные системы, среды и оболочки: Учебное пособие/ Т.Л.Партыка, И.И.Попов. – М: Форум, 2018.