1.1 Основные понятия об операционных системах

Операционная система(ОС)- это набор взаимодействующих программ (часто называемых системными) и информационных структур данных. с помощью которых организуется взаимодействие пользователя и прикладных программ с вычислительной системой.

Особенности ОС по сравнению с другими программами:

* исключено тесное взаимодействие с аппаратным уровнем:
* управление выполнением других программ:
* исключена большая частота использования:
* очень сложная и объемная программная реализация:
* высокие требования к надежности и скорости работы

**Ресурсы**

|  |  |
| --- | --- |
| физические | логические |
| (основная память, ЦП, внешняя память, устройства ввода/вывода данных) | (файл, программные модули, внутренние информационные структуры, различные системные объекты) |

1.2 Основные этап развития ОС

**Этап 1-й:50-е годы ХХ века**

Появляются первые **управляющие программы-мониторы** как прообраз будущих ОС

**Этап 2-й: 60-е годы**

Создание компанией IBM первого семейства универсальных ЭВМ широко использования - семейства **IBM/360**, для этого семейства было разработано несколько версий ОС с общим названием **OS/360**. В этих ОС была реализована концепция **многозадачности**

**Этап 3-й: 70-е годы**

Широкое распространение **мини-компьютеров**, лидером являлась марка **PDP/VAX**

Реализация многопользовательского режима работы

Создание **ОС UNIX**

**Этап 4-й: 80-е годы**

Начало эры персональных компьютеров. Базовой ОС для ПК стала простая система **DOS**(Disk Operating System).

Создана система **MacOS** для ПК Nacintosh фирм Apple.

Для платформы Intel в конце 80-ч совместно с компаниями IBM и Microsoft была создана мощная многозадачная и многопользовательская система **OS/2**.

Начало развития сетевых технологий

**Этап 5-й: 90-е годы**

Распространение по миру ОС семейства **Windows** - от версии Windows 95 до Windows ME(Millenium Edition). Это семейство распространилось и на сектор сетевых ОС в виде "тяжеловесной" системы NT(т.е. New Technology) и ее более поздней реализации **Windows 2000**(версии Professional, Server, Advancer Server, Datacenter).

Появление **Linux**(автор- Линус Торвальдс).

**Этап 6-й: первое десятилетие нового века**

Развитие семейства Windows как в пользовательском, так и серверном варианте. Выпущены версии **XP**, **Vista,** **W7**. Основные серверные версии - Server 2003 и Server 2008.

Наиболее известные версии(дистрибутивы) Linux - **Ubuntu, Mandriva, ASP, OpenSUSE, Debian.**

1.3 Основные типы и функции ОС

1. **Универсальные пользовательские системы** для массовых ПК.

*Основное назначение* - создание удобной среды для работы пользователя с множеством прикладных программ в условиях многозадачности.

2. **Универсальные серверные системы**

*Назначение* - управление работой корпоративных сетей: взаимодействие с клиентскими системами, управление пользователями с разграничением прав, обеспечение безопасности данных.

3.Системы для **портативных и мобильных устройств**, вынужденные работать в условиях очень ограниченных ресурсов.

4. **Специализированные системы**, в первую очередь - системы реального времени с необходимостью максимально быстрого реагирования на внешние события(управление быстротекущими процессами). Пример: **QNX.**

Основные функции универсальных систем общего назначении:

1. Рациональное распределение физических и логических ресурсов между различными приложениям

*Включает в себя:*

* распределение времени ЦП между выполняющимися приложениями (управление **процессами** и **потоками**):
* распределение ОП между выполняющимися приложениями в условиях ограниченного объема (управление **памятью**)
* выделение приложением необходимых внешних устройств и организация **взаимодействия** приложений с устройствами с обязательным контролем со стороны системы(управление **устройствами**)
* распределение внешней памяти и организация хранения информации с помощью файлов (управление **файлами**)

2. Поддержка сетевых взаимодействий на основе стандартов и протоколов

3. Обеспечение безопасности хранения данных.

4. Предоставление разработчикам и приложениям широкого спектра

системных функций уровня **АРI** (Application Program Interface — интерфейс прикладного программирования). Именно через эти функции приложения получают доступ ко всем ресурсам ВС. НаборWin32 АРI включает около 2000 функций.

**2.1. Понятие процессах и потоках**

**Процесс** — системный объект, который соответствует запущенному на

выполнение приложения.

В многозадачных системах обычной является ситуация существования

одновременно нескольких процессов (**внешняя многозадачность**).

**Внутренняя многозадачность** возникает, когда в одном процессе

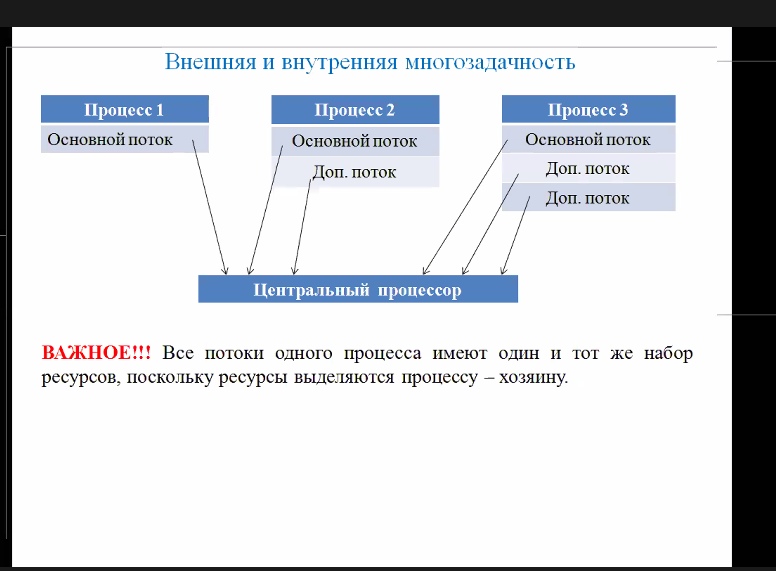
создается хотя бы два так называемых потока.

**Поток выполнения (thread)** — это относительно самостоятельная часть

исполняемого программного кода приложения. Самостоятельность потока, как системного объекта определяется тем, что именно поток претендует на время ЦП. Поток — это единица распределения времени ЦП.

**Два типа приложений:**

* Однопоточные. Состоят только из 1 потока, который создается системой автоматически и содержит весь исполняемый код приложения.
* Многопоточные. Кроме главного потока существуют и могут запускаться дополнительные (вторичные) потоки.



Следствие:

1. Создание нового процесса — достаточно трудоемкий труд, тогда как

создание новых потоков внутри уже существующих процессов

выполняется гораздо быстрее.

2. Все потоки одного процесса используют одну и ту же область памяти,

выделенную их процессу хозяину. Поэтому потоки имеют

неограниченный доступ к области памяти, используемым другими потоками процесса.

Следствие (из 2)

+ Очень легко организуется обмен данными между потоками процесса.

— Неправильная работа одного потока может «подвесить» остальные потоки и процесс в целом! Необходима обязательная отладка многопоточных приложений.

Управление процессами и потоками, включающими в себя решение следующих основных задач:

* **создание** и **уничтожение** процессов и потоков;
* **планирование** порядка выполнения потоков
* **синхронизация** и **взаимодействие** потоков

**2.2 Создание и уничтожение процессов и потоков**

Создание процесса, как системного объекта:

**1. Формирование** специальной системной структуры данных с полной

информацией о запускаемом приложении — дескриптор( описатель) процесса и включение этой структуры в системную таблицу дескриптора процессов.

**2. Создание** системного объекта « Основной поток», включающий формирование еще одной структуры данных с информацией о потоке дескриптор потока.

**3. Загрузка** исполняемого кода в память с созданием необходимых таблиц.

**4. Включение** главного потока в число претендентов на время ЦП.

Структура дескрипторов процесса

* уникальный идентификатор процесса:
* время создания процесса;
* полное имя исполняемого файла с кодом приложения;
* выделяемые процессу ресурсы
* указатель на структуру — описатель фонового потока процесса.

Структура дескрипторов потока

* уникальный идентификатор потока;
* приоритет потока( используется при планировании порялка выполнения потоков);
* текущее состояние потока;
* дополнительные параметры для планировщика потоков;
* указатель на дополнительную структуру — контекст потока, который
* используется для сохранения текущего состояния потока в момент прерывания его выполнения

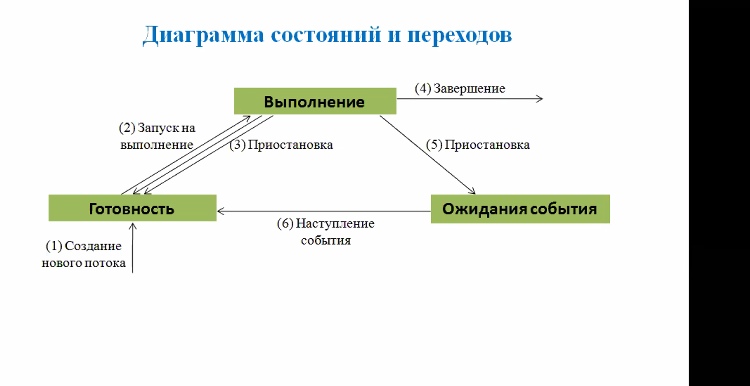


Варианты завершения потоков:

* нормальное
* аварийное
* принудительное

В этих случаях происходит удаление всей связанной с потоком информации. При завершении главного потока происходит удаление и информации о процессе в целом.





**2.3 Состояние потоков и переходы между состояниями**

Созданные системой потоки, за время своего существования могут находится в одном из 3-х состояний:

- состояние "выполнение"- код потока выполняется процессором

- состояние "готовность к выполнению" - поток готов продолжить свое выполнение, но процессор пока занят выполнением другого потока

- состояние "ожидание события" - поток НЕ готов продолжать свое выполнение, пока не произойдет некоторое событие, такие блокированные потоки НЕ претендуют на время ЦП.

**2.4 Планирование выполнения потоков**

Если в системе существует более одного готового к выполнению потока, возникает задача определения очередности предоставления им процессора — задача планирования. Для ее решения в ядре системы создается специальный модуль — планировщик: потоков.

Разработка эффективного планировщика — непростая задач: он должен быть достаточно простым и в то же время обеспечивать эффективную работу пользователя, справедливое обслуживание всех потоков и рациональную загрузку оборудования.

**Основной принцип** — вытесняющая многозадачность на основе квантования времени ЦП и динамическое изменение приоритетов потоков.

Квант — интервал времени, в течение которого поток может выполняться процессором

Квантование — основной способ создания иллюзии одновременного выполнения нескольких программ единственным процессором.

Как выбрать величину кванта?

Большие кванты (секунды и более) приводят к потере иллюзии одновременного выполнения нескольких потоков, особенно для мультимедийных потоков.

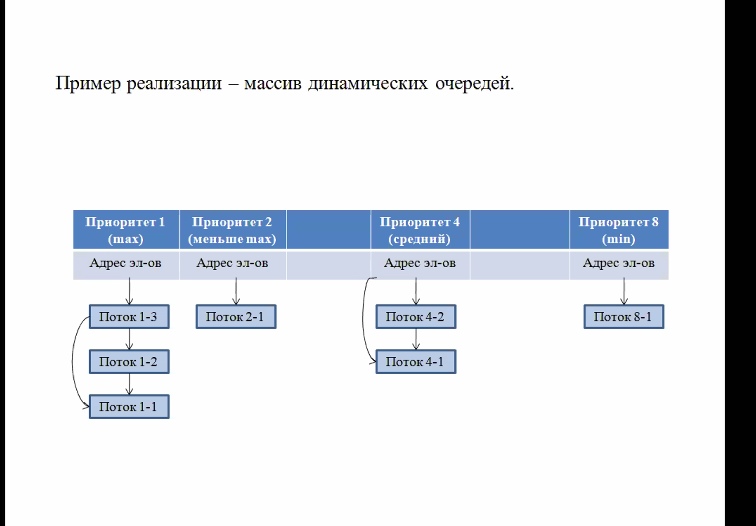
Очень маленькие кванты (микросекунды =10-6 сек) приводят к очень частым переключениям между потоками, что плохо, поскольку каждое переключение требует некоторого времени на сохранение состояния прерываемого потока и возобновление активизирующего потока.

Во многих системах приоритеты разделена на три группы:

* приоритеты сверхважных потоков (потоки реального времени);
* приоритеты системных потоков
* приоритеты прикладных потоков

Динамические изменения приоритетов в зависимости от поведения потоков:

* Длинные потоки полностью используют выделяемые канты и их приоритеты система уменьшает.
* Короткие потоки, наоборот, НЕ используют весь выделенный квант, переходя в заблокированное состояние и досрочно освобождая процессор для других готовых потоков.



**Планирование потоков**

* Находится 1-ая непустая очередь и на выполнение запускается поток, стоящий в ней на 1-ом месте
* Когда активный поток прерывается, он включается в конец очереди для его значения приоритета.

**Моменты активизации планировшика**

* завершение канта времени;
* завершение (нормальное или аварийное) текущего активного потока;
* переход активного потока в заблокированное состояние
* появление более важного готового потока.
* Возникновение события, приводящего к разблокированию одного или нескольких потоков.

**2.5 Взаимодействие потоков**

**Взаимодействие потоков включает:**

* обмен данными между потоками
* синхронизация выполнения потоков

**Системные средства обмена:**

* каналы(pipe): достаточно медленно, но надежно
* общие(разделительные, shared) области памяти: быстро, но опасно

**Проблема синхронизации работы потоков**: два потока одновременно должны вносить изменения в одну и ту же структуру

**Решение** - использование системных средств синхронизации:

критические секции для потоков в 1-ом процессе;

мьютексы для потоков в разных процессах;

семафоры - контроль доступа к набору общих однотипных потоков

*Пусть 2 потока должны последовательно обрабатывать две общие структуры.* **Тогда возможна следующая ситуация:**

* поток 1 начал обработку структуры 1, блокировал ее и был прерван планировщиком;
* поток 2 начал обработку структуры 2, блокировал ее и был прерван планировщиком;
* поток 1 возобновляет свою работу, запрашивает структуру 2, но она блокирована потоком 2 и поэтому поток 1 опять блокируется
* поток 2 возобновляет свою работу, запрашивает структуру 1, но она блокирована потоком 1 и поэтому поток 2 опять блокируется

Результат - оба потока не могут продолжить свою работу.

Решение - отслеживание системой времени пребывание потоков в заблокированном состоянии, и в случае превышения этим в решениях некоторой заданной пороговой величины - принудительное освобождение заблокированных ресурсов

**Пример критической секции**

**В коде программы выделяется критический фрагмент, который непосредственно выполняет обработку общей структуры данных, система:**

* проверяет значение своей внутренней логической переменной(*флаг* "занято/свободно")
* если значение *флага* - "свободно", то система изменяет это значение на "занято" и разрешает программе вход в критический фрагмент
* в противном случае, система блокирует поток, приостанавливая его дальнейшее выполнение

**Когда выполнение программы дойдет до конца, система:**

* изменяет значение *флага* на "свободно";
* просмотрит список заблокированных потоков и переведет их из состояния готовности к выполнению

**3.2 Страничная организация памяти**

Физическая страница (ФС) - фиксированный по размеру

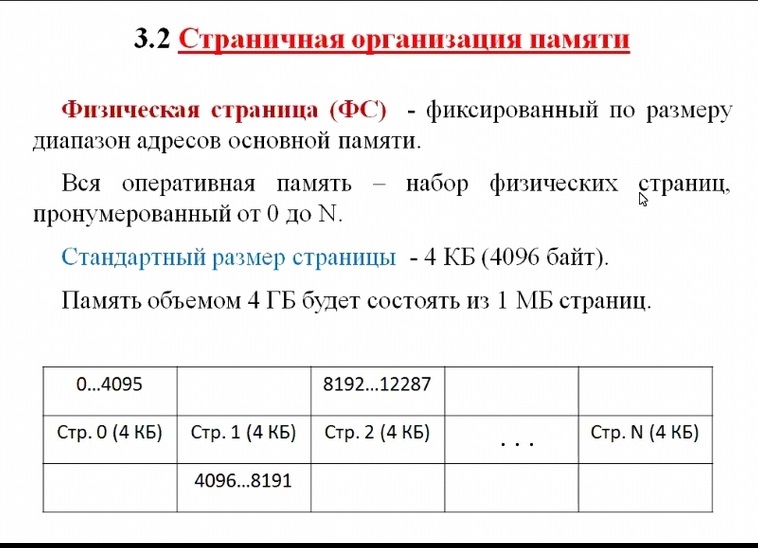
диапазон адресов основной памяти.

Вся оперативная память — набор физических страниц.

пронумерованный от 0 до N.

Стандартный размер страницы - 4 КБ (4096 байт).

Память объемом 4 ГБ будет состоять из 1 МБ страниц.



Виртуальная (логическая) странница — аналогичный по

размеру (4 КБ) диапазон адресов Виртуального Адресного

Пространства (ВАП),

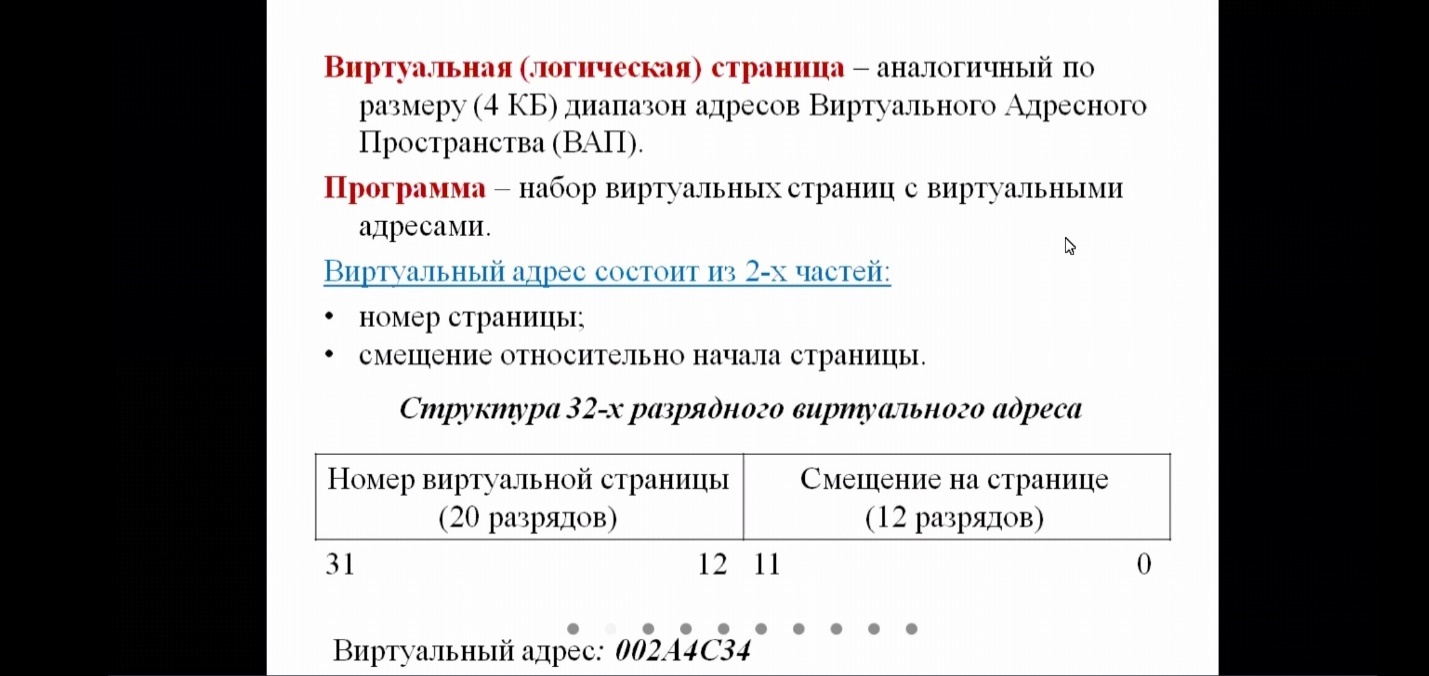
Программа — набор виртуальных страниц с виртуальными

адресами

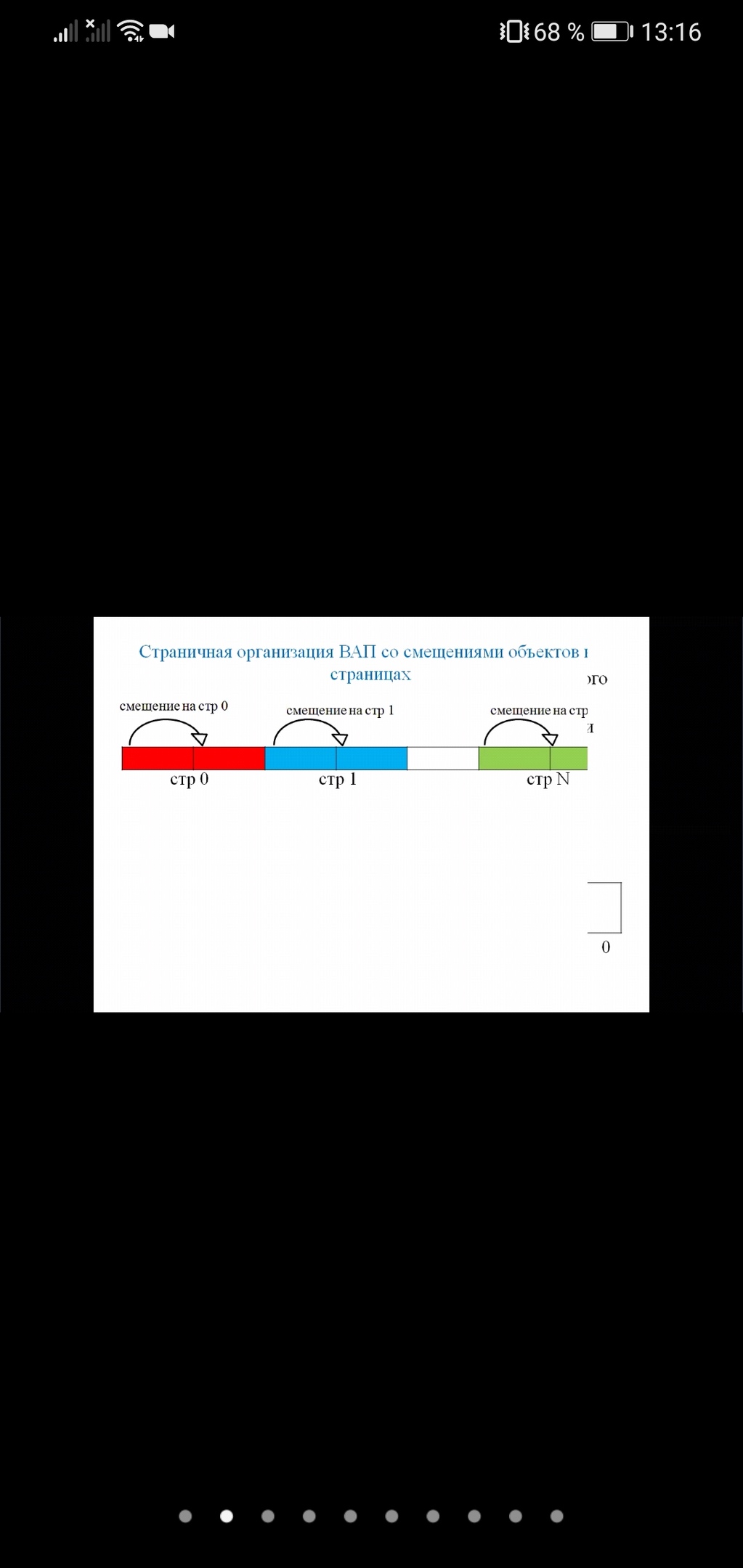
Виртуальный адрес состоит из 2-х частей:

* номер страницы;
* Смещение относительно начала страницы.

Структура 32-х разрядного виртуального адреса



Виртуальный адрес: 002А4С34



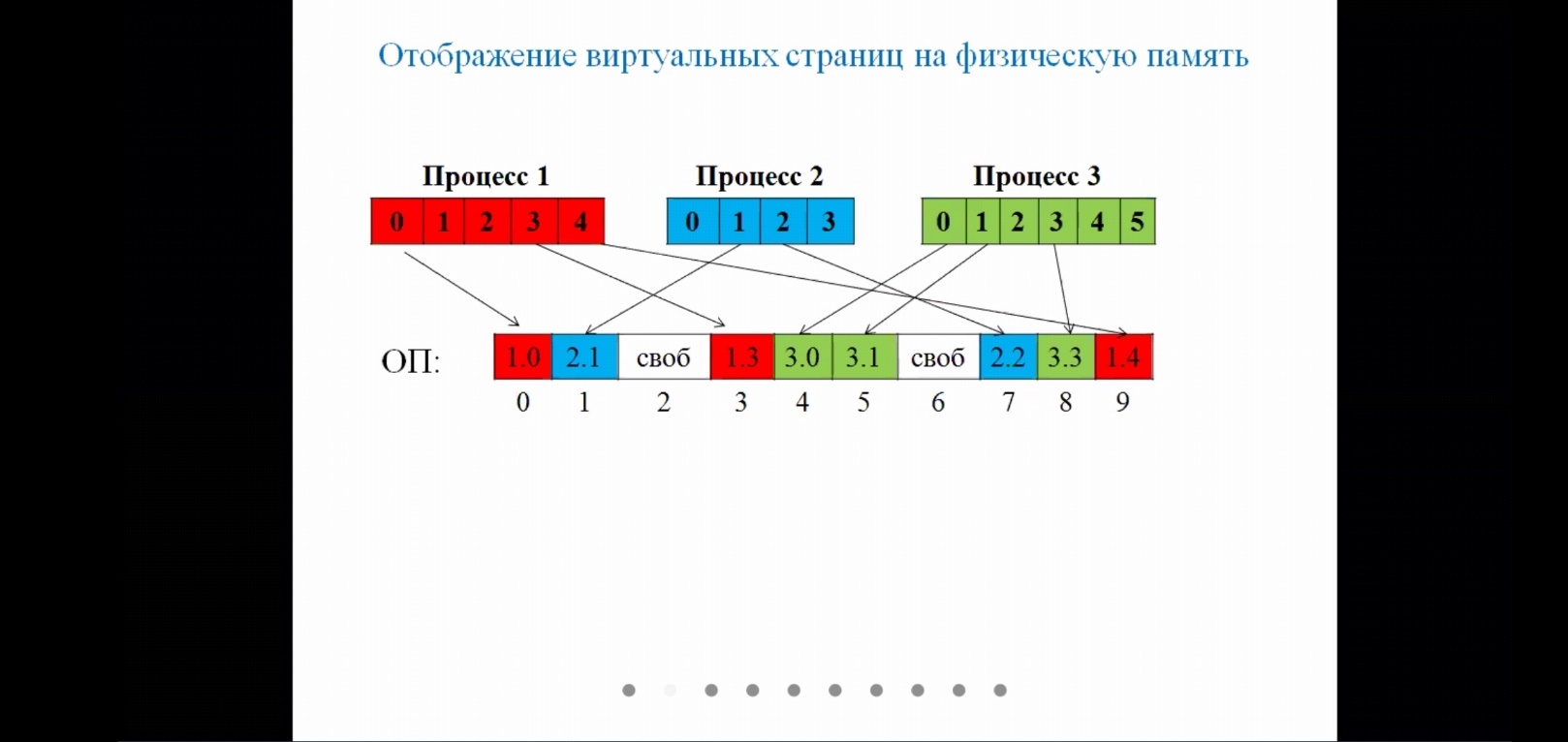


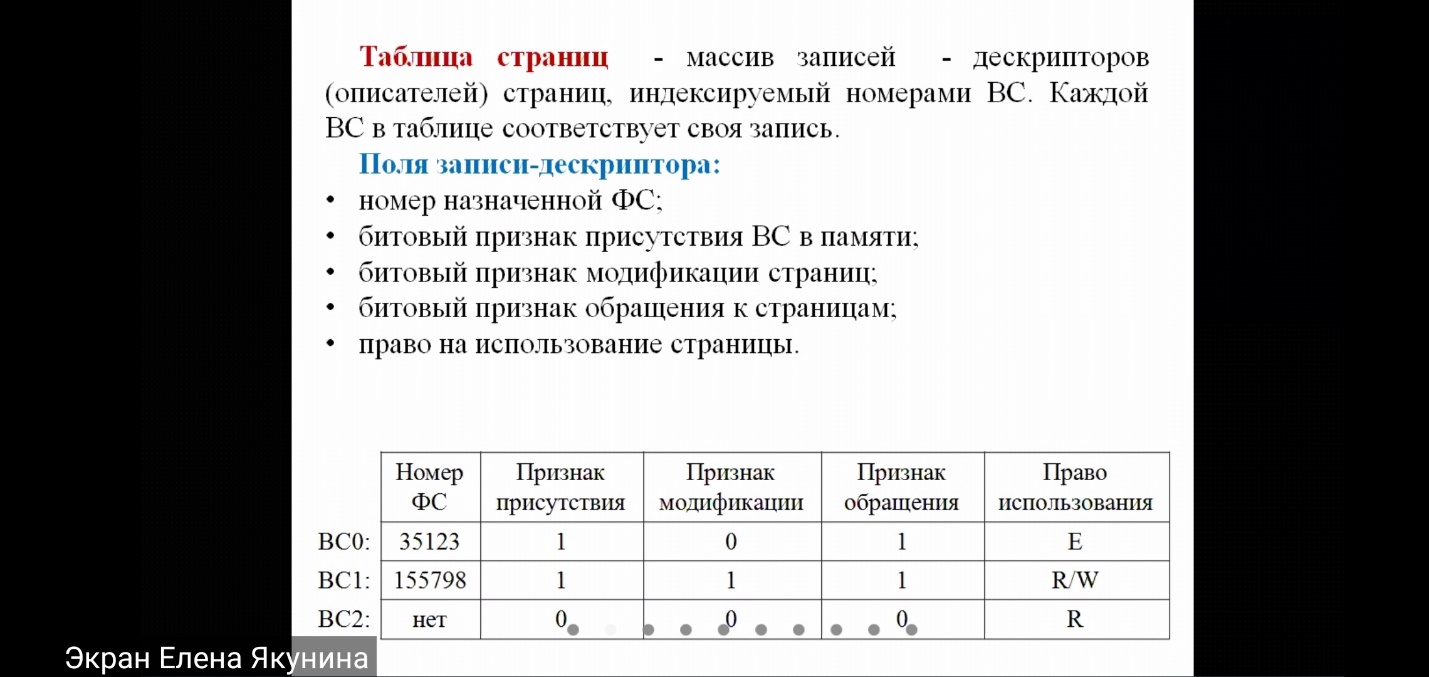
Таблица страниц - массив записей - дескрипторов

(описателей) страниц, индексируемый номерами ВС. Каждой

ВС в таблице соответствует своя запись.

Поля записи-дескриптора:

* номер назначенной ФС
* битовый признак присутствия ВС в памяти
* битовый признак модификации страниц
* битовый признак обращения к страницам
* право на использование страницы.



Таблицы страниц:

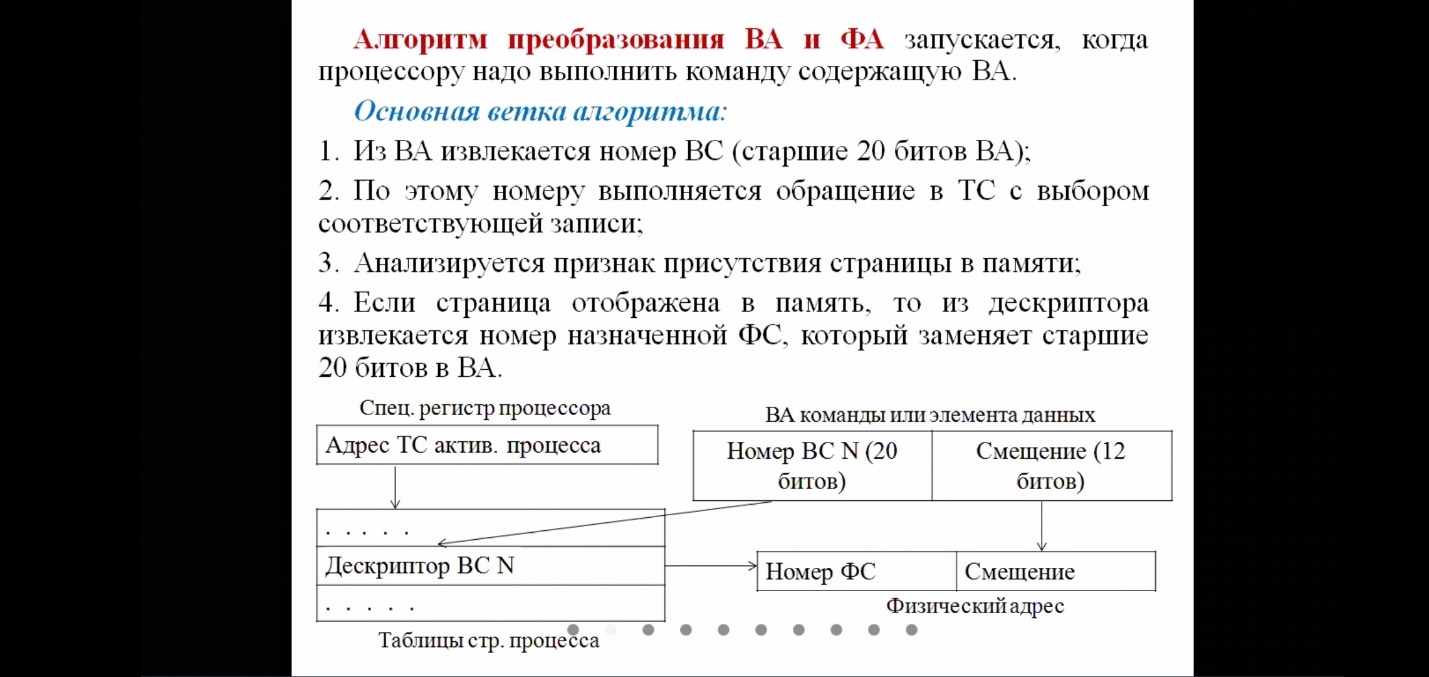
* создаются при создании системой объекта «процесс», на то, что входит некоторое время;
* все потоки одного процесса используют одну и ту же ТС (одно адресное пространство);
* выполняются системными структурами и размещаются в системной области памяти;
* каждый процесс «знает» только свою ТС. доступа к «чужим» таблицам у процесса нет (защита адресных пространств процессов):
* при переключении процессов происходит и переключение таблиц (адрес размещения в памяти таблицы приостанавливаемого процесса, сохраняется в контексте).

Алгоритм преобразования ВА и ФА запускается. когда

процессору надо выполнить команду содержащую ВА.

Основная ветка алгоритма:

1. Из ВА извлекается номер ВС (старшие 20 битов ВА):
2. По этому номеру выполняется обращение в ТС с выбором соответствующей записи
3. Анализируется признак присутствия страницы в памяти
4. Если страница отображена в память, то из дескриптора извлекается номер назначенной ФС, который заменяет старшие 20 битов в ВА.



Если запрашиваемой страницы в памяти нет, то:

генерируется специальное событие — ошибка «Отсутствие страниц»:

активный процесс блокируется (переходит в состояние ожидания), что дает возможность процессору переключиться на выполнение других процессов (потоков)

1. выполняется поиск запрашиваемой страницы во внешней памяти (для ускорения поиска используется специальный буфер страниц):
2. для найденной страницы определяется свободная ФС (ситуация отсутствия свободных ФС обрабатывается отдельной веткой алгоритма);
3. выполняется корректировка записи-дескриптора для запрошенной
4. страницы:

* заносится номер назначенной ФС
* устанавливается признак присутствия
* устанавливается все остальные признаки

1. содержимое ВС загружается в память в назначенную область:
2. остановленный процесс разблокируется, активизируется и возобновляет работу с прерванной команды, для которой теперь уже можно преобразовать ВА в ФА стандартным образом.

А если при загрузке нет ни одной чыободной ФС?

Запускается специальный алгоритм поиска «жертвы». т.е. занятой ФС. которую приходится освобождать

Используемые принципы определения «жертвы»:

* выбрать самую редко использующую страницу (по числу обращений);
* выбрать самую «древнюю» страницу` к которой не было обращений дольше остальных.

После определения освобождаемой страницы:

* проверяется необходимость сохранения ее на диске(изменилось ли ее содержимое);
* признак присутствия изменяется на «выгружена»;
* освободившиеся ФС назначаются затребованным ВС с выполнением необходимых операций.

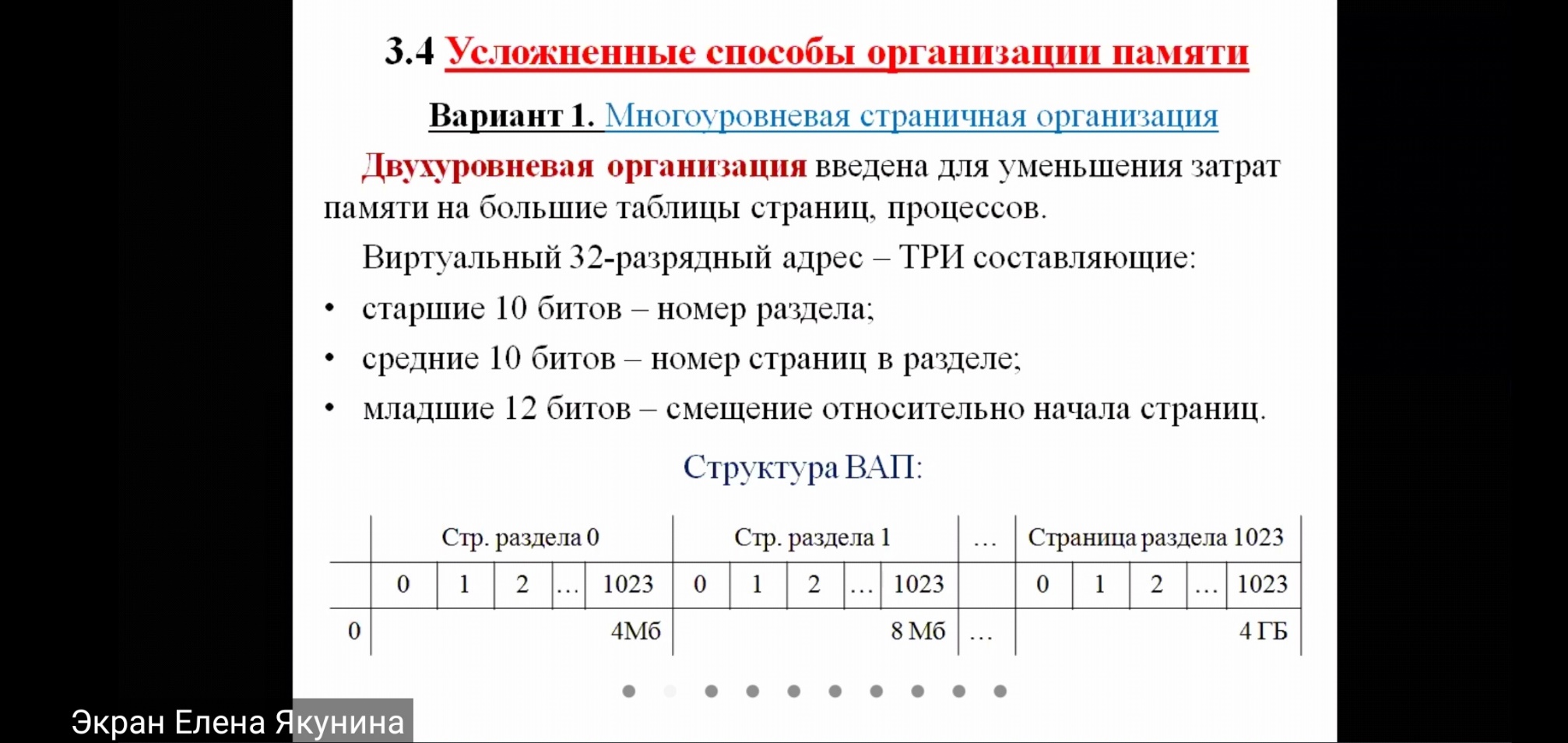
**3.4** **Усложненные способы организации памяти**

**Вариант 1.** Многоуровневая страничная организация

Двухуровневая организация введена для уменьшения затрат памяти на большие таблицы страниц, процессов.

Виртуальный 32-разрядный адрес – ТРИ составляющие:

* Старшие 10 битов – номер раздела
* Средние 10 битов – номер страниц в разделе
* Младшие 12 битов – смещение относительно начала страниц



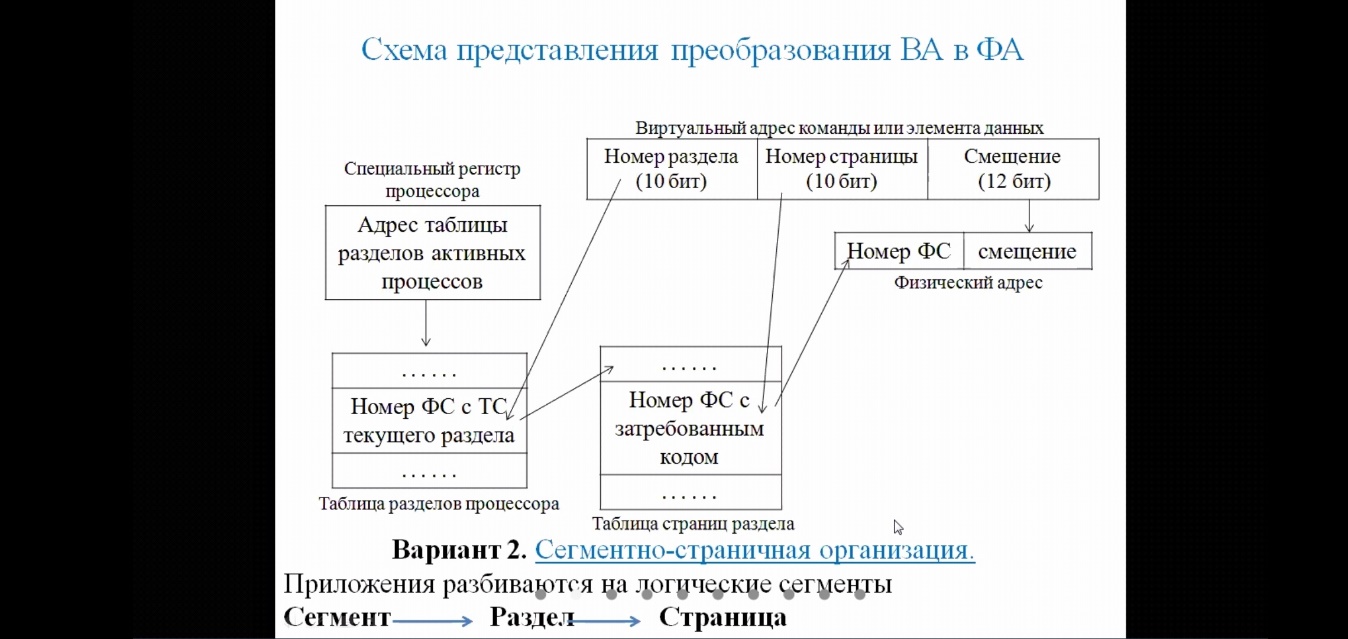
Каждая ТС занимает ровно одну физическую страницу.

Таблица разделов – набор 1024 записей. Каждая запись содержит информацию о соответствующей ТС – загружена ли она в память и в какую ФС, как часто используется.

Каждая из 1024 ТС имеет обычную структуру и содержит данные от 1024 физических страницах, выделенных процессору.

Преобразование ВА в ФА:

1. Выделяется номер раздела и выбирается запись в таблице разделов.
2. Проверяется наличие в памяти страницы с запрошенной таблицей страниц.
3. Если ТС нет в памяти, выполняется её поиск и загрузка обычным образом ( с приостановкой процессора и возможным вытеснением страниц)
4. Из ВА выделяется номер ВС и в соответствующей ТС выбирается соответствующая запись.
5. Запись обрабатывается обычным образом



**Вариант 2.** Сегментно-страничная организация

Приложения разбиваются на логические сегменты

Сегмент 🡪🡪🡪 Раздел 🡪🡪🡪 Страница

**4.1 Общая структура подсистемы УВУ**

**Назначение**-изоляция прикладных программ от аппаратных особенностей конкретных технических устройств ввода-вывода данных.

Основной принцип построения - **многоуровневый.**

*Уровень 1.* **Аппаратно-зависимый:** сами устройства (вместе с контроллерами) и программы-драйверы для управления ими.

*Уровень 2.* **Аппаратно-независимый:** основные управляющие программы и соответствующие структуры данных.

*Уровень З.* **Системные вызовы:** через которые приложения могут запрашивать выполнение операций ввода-вывода.



п

**4.2 Низкоуровневое управление устройствами**

На аппаратном уровне - **контроллеры** (электронные схемы управления конкретным устройством).

Па программном уровне - **драйверы** устройств.

Взаимодействие драйвера с устройством через регистры контроллеров. Базовый набор регистров:

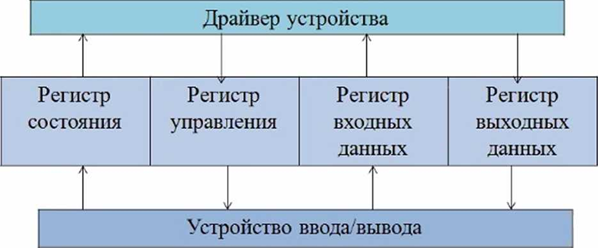
**Регистр состояния устройства** - (свободно занято, готово или нет успешно или нет) выполнения очередной команды.

**Регистр управления (командный регистр)** - в него драйвер заносит очередную команду, которую должно выполнить устройство.

**Регистр входных данных** - куца устройство заносит очередную порцию данных (1-4 байта) и откуда они извлекаются драйвером.

**Регистр выходных данных,** в который драйвер заносит очередную порцию данных для обработки их устройством.

**Взаимодействие программы-драйвера с регистрами контроллера**

****

Все регистры всех установленных устройств перенумерованы и называются портами ввода-вывода

Для взаимодействия с портами в системе команд процессора предусмотрены специальные команды:

* **Команда (OUT N)** заносит байты в порт с номером N.
* **Команда (IN N)** позволяет прочитать байты и порта с номером N.

Команды - привилегированные, т.е. могут выполняться только в режиме ядра! Эти команды составляют основу программного кода драйверов.

Внутренние прерывания

Возникают «внутри» процессора при возможности выполнения очередной команды активного процесса.

Возможные причины:

* попытка деления на ноль;
* попытка обращения по неправильному адресу:
* попытка выполнения запрещенной команды в пользовательском режиме;
* попытка обращения к странице, отсутствующей в памяти.

Такие особые ситуации называются исключениями (exception) и для их обработки предусмотрены специальные механизмы.

Программные прерывания (псевдо прерывания) возникают как результат выполнения специальных команд в программном коде.

**Работа механизма прерываний**

Таблица векторов прерываний

****

Обработка прерываний на программном уровне с **помощью программ-обработчиков.**

Быстрый запуск обработчиков - с помощью специальных **таблицы векторов прерываний.**

**Таблица** - набор 256 записей-адресов размещения в памяти программ-обработчиков.

Вход таблицу - по индексу-номеру прерывания.

Размер таблицы 256\*4= 1Кбайт

**Размещение** - фиксированное в самом начале ОП (первый килобайт).

**Контроллер прерываний** по номеру прерывания определяет в таблице адрес размещения в памяти соответствующего обработчика и тем самым инициирует обработку прерывания.

Сигналы наступают на контроллер прерываний, который:

* распознает тип прерывания и его приоритет;
* проверяет состояние процессора и если он свободен, то сразу запускает специальную программу обработчик прерываний:
* если процессор занят, то контроллер сравнивает приоритеты нового прерывания и активного потока;
* если приоритет активного потока меньше приоритета новою прерывания, то активный поток досрочно приостанавливается и запускается обработка нового прерывания;
* в противном случае (когда процессор занят очень срочной работой) контроллер сохраняет сигнал о возникновении прерывания до тех нор. пока процессор не занесет выполнение срочной работы.

Упорядоченность прерываний по их важности (приоритету):

* критические сбои аппаратуры;
* прерывание от таймера:
* прерывание от дисковых устройств;
* прерывание от сетевых устройств:
* прерывания от клавиатуры и мыши.

Обработка прерываний на аппаратном уровне с помощью

контроллера прерываний.

Все прерывания пронумерованы от 0 до 255 (от 0016 до

FF16 = 1111111116).

Каждому прерыванию соответствует электрический сигнал, генерируемый устройством.

**4.3 Прерывание и их обработка**

Прерывание — сигнал системе с некоторым событием.

Основные типы:

Внешние или аппаратные прерывания;

Внутренние прерывания:

Программные прерывания (псевдопрерывания).

* Важнейшие внешние прерывания. Возможные источники:
* системный таймер (событие - завершение кванта времени);
* контроллер дисковода (событие - освобождение устройства);
* клавиатура (событие — нажатие клавиши):
* мышь (событие - нажатие кнопки);
* сетевая плата (получение порции информации).

**4.4 Высокоуровневое управление устройствами**

Основное назначение модуля «**Диспетчер ввода-вывода**» - трансляция

вызовов операций ввода-вывода от прикладных программ драйвером и

обратно.

Основная информационная структура — Таблица Устройств как набор

**записей-дескрипторов**, содержащих информацию о каждом устройстве:

* тип устройств и его характеристики;
* номера используемых портов;
* номера назначенных устройству прерываний;
* информацию об используемых устройством драйверах;
* текущее состояние устройства;
* идентификатор процесса, использующего устройство в данный

момент;

* адрес начала списка потоков, ожидающих освобождения данного

устройства;

* адреса размещения в памяти вспомогательных буферов для обмена

данными;

* логическое имя устройства, используемое высокоуровневыми

системными вызовам.

Программный код Диспетчера реализует следующие

операции:

1. Обработку системного вызова с проверкой правильности

передаваемых в вызове параметров.

2. Проверку состояния запрашиваемого устройства и в случае его

занятости:

- Перевод активного потока в блокированное состояние с

включением его в список блокированных потоков.

- Вызов планировщика потоков для активизации другого потока.

3. Если устройство свободно, то Диспетчер:

- Вызывает соответствующий драйвер и передает ему

необходимую информацию.

- Уступает процессор запущенному драйверу.

4. После выполнения драйвером необходимых агрегаций,

Диспетчер активизируется, выполняет анализ полученной от

драйвера информации и возвращает результат выполненной

операции ввода-вывода потоку-источнику запроса.

**Основные концепции использования файлов:**

1. Файлы имеют имена;

2. Файлы образуют иерархическую древовидную структуру;

3. Специальные файлы -— каталоги, предназначены для

группирования других файлов. в том числе и каталогов;

4. Основные типы файлов: текстовые. графические, звуковые,

исполняемые.

5. Файлы имеют атрибуты

**Основные задачи файловой системы:**

1. Распределение внешней памяти между файлами.

2. Учет свободной и занятой внешней памяти.

3. Преобразование внешних имен файлов в адреса размещения,

соответствующих данных во внешней памяти.

4. Выполнение основных операций с файлами: создание и

удаление, открытие и закрытие, чтение и запись из файла,

копирование и перемещение, переименование.

Основные типы файловых систем:

1. Для семейства Windows - различные версии NTFS(

и иногда FATЗ2, FAT64.

2. Для семейства UNIX-LINUX - различные версии системы

ufs, такие как ехt2 и ехt3.

**5.2 Физическая организация хранения информации**

Основной тип устройств внешней памяти -— жесткие

магнитные диски.

**Устройство** — одна или несколько пластин на одной оси,

вращающихся с высокой скоростью (около 10 тыс. оборотов в

минуту).

**Чтение/запись** — как правило с обеих сторон (поверхностей)

этих пластин с помощью гребенки магнитных головок.

**Перемещение** — дискретное с очень малым шагом/уза счет

чего на поверхности КАК БЫ образуются концентрические

дорожки.

Каждая дорожка с помощью специальных электронных

маркеров разбивается на отдельные участки — секторы.

Стандартный размер физического сектора - 512 байт.

**Сектор** — основная единица обмена данными с дисками:

прочитать или записать можно только весь сектор. но не его

часть: !!

Физический адрес сектора задается тремя числами:

* номер используемой поверхности (т.е. номер головки в блоке

головок):

* номер дорожки на поверхности;
* номер сектора на дорожке.

Контроллеры современных дисководов позволяют выполнять

сквозную нумерацию секторов: все секторы всех дорожек и

поверхностей пронумерованы от 0 до N.

Кластер-блок (NTFS) — это последовательность нескольких

соседних секторов, рассматриваемая файловой системой как

единое целое.

Минимальный блок - 2 сектора, т.е. 1 КБ дисковой памяти.

Стандартный блок — 8 секторов, т.е. 4 КБ дисковой памяти.

Небольшие кластеры:

+ позволяет достаточно экономно использовать дисковую

память. особенно для очень маленьких файлов (несколько КБ)

- Увеличивают размеры структур данных — (таблиц),

необходимых для управления большим числом небольших

блоков.

- Замедляют обработку больших файлов. разбросанных по

разным (не смежным) блокам дисковой памяти.

Большие кластеры:

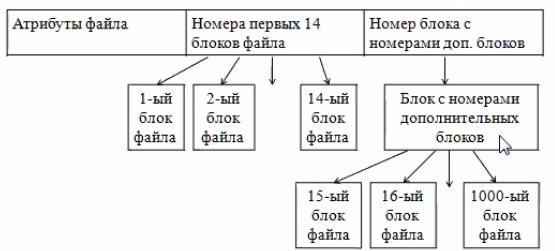
+ уменьшают накладных расходов на хранение информации об

используемой дисковой памяти

+ Ускоряют обработку больших файлов

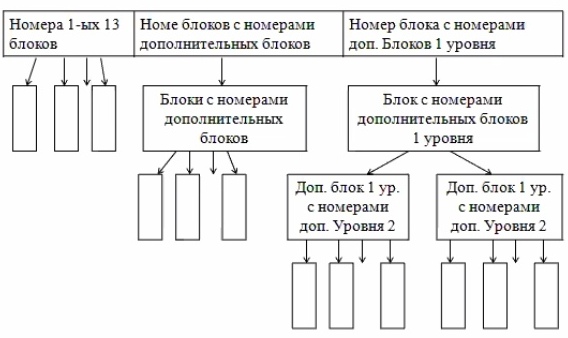
- Увеличивают фрагментацию памяти и не производные потери

памяти, особенно на маленьких файлах.

**Индексный узел для больших файлов:**

**Индексный узел для очень больших файлов (адресная**

**часть)**

****

Можно использовать третий уровень адресации. что

позволяет обрабатывать файлы до 4 ТБ.

Недостатки такой схемы - некоторое замедление

обработки очень больших файлов ввиду необходимости

дополнительных операций с дополнительными блоками (их

надо найти на диске, прочитать в память и обработать).

Все индексные узлы перенумерованы целыми числами и

являются системными ресурсами/

Файловая система должна вести учет свободных и занятых

индексных узлов.

Если все узлы заняты, то создать новый файл невозможно!

**5.4 Реализация основных операций с файлами**

Одна из задач файловой системы - связь внешнего имени

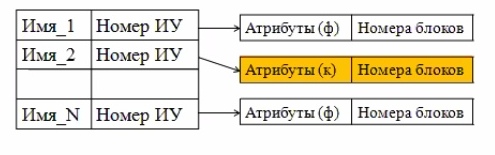
файла с его параметрами и размещением на диске.

Каталог — массив записей (таблица). каждая из

которых имеет всего два поля:

* универсальное символьное имя файла;
* номер выделенного файлу индексного узла.

Структура каталога



Вспомогательные операции - открытие и закрытие

файлов.

Открытие — подготовка файла к использованию, поиск,

загрузка ИУ, запись информации в файле в специальной

таблице открытых файлов для быстрого доступа к файлу при

выполнения операций чтения и записи.

Закрытие - удаление информации с файла в таблице

открытия файлов.