

## Страничная организация памяти

---

Организация памяти в виде страниц борется с двумя проблемами

- Внешней фрагментацией – используются блоки фиксированного размера в виртуальной и физической памяти, т.е. все запросы на выделение памяти будут кратны, не будет оставаться некратных зон.
- Внутренняя фрагментация – блоки достаточно малого размера, поэтому (K) будет мал. (K) - это есть **внутренняя фрагментация** – фрагментация внутри блока.

# Страничная организация памяти

---

## **С точки зрения программиста:**

- Процессам виртуальное адресное пространство предоставляется непрерывным, от байта 0 до байта N
- N зависит от аппаратной поддержки (например 32бит. - адр.пространство 4Гб), делится соответственно.
- В реальности виртуальные страницы распределены по страницам физической памяти далеко не непрерывно и не один к одному. Это два разных мира – физические страницы и виртуальные страницы. Это ключевой аспект, который надо понимать.

# Страничная организация памяти

---

## **С точки зрения менеджера памяти(+):**

- Эффективное использование памяти из-за очень низкой внутренней фрагментации
- Внешняя фрагментация полностью отсутствует и не нужно дефрагментировать

## **С точки зрения защиты(+):**

- Процесс имеет доступ только к своему адресному пространству.

Допущение – все страницы виртуальной памяти всегда находятся в страницах физической памяти. Не будем думать, что есть только виртуальные страницы, а физических – их нет, т.е. полное отображение виртуальной и физической памяти.

Предположим, что все страницы резидентно в памяти, это необходимо, чтобы понять как работает **трансляция адресов**.

# Трансляция адресов

---

## **Трансляция виртуального адреса:**

Виртуальный адрес состоит из двух частей: номер виртуальной страницы (VPN) и смещение внутри страницы

**Номер виртуальной страницы (VPN** - virtual page number) это индекс в таблице страниц (Pagetable)

**Запись в таблице страниц (PTE** – page table entry) содержит номер фрейма (**PFN** –page frame number)

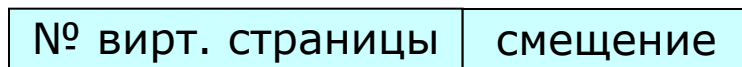
**Номер фрейма** – это номер физической страницы.

**Фрейм** – это страница физической памяти.

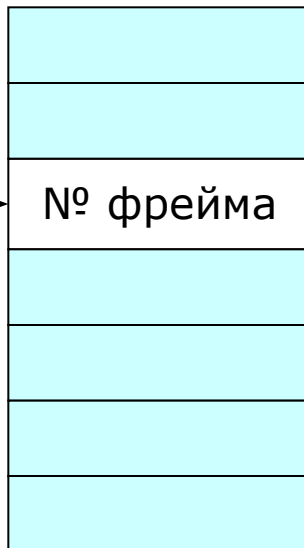
Смысл таблицы страниц – одна запись в таблице страниц (PTE) на одну страницу виртуального адресного пространства (VPN), отображает VPN на PFN. Какая **виртуальная страница** соответствует какому **фрейму физической** памяти.

# Трансляция адресов

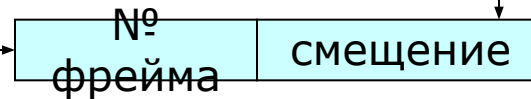
**Виртуальный адрес**



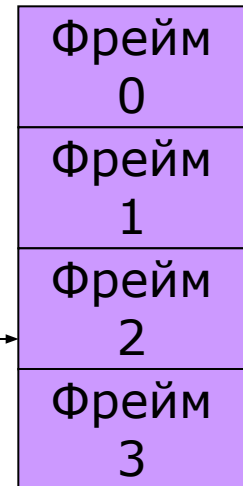
**Таблица страниц**



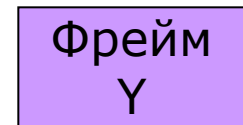
**Физический адрес**



**Физическая память**



...



# Трансляция адресов

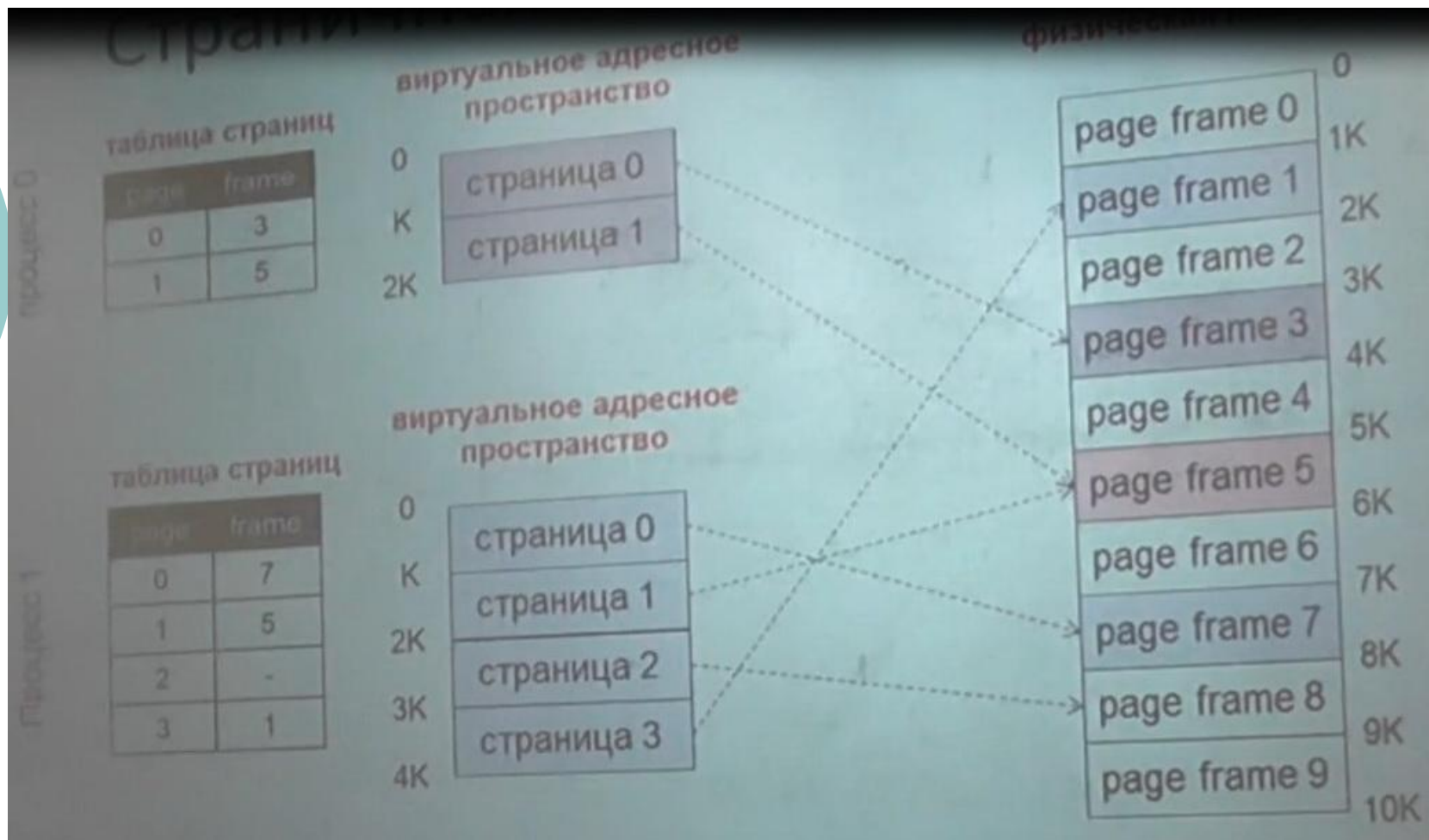
---

Есть виртуальный адрес, состоящий из двух частей:

- 1) №вирт.страницы, по нему идет поиск в таблице страниц и находится № фрейма, он составляет одну часть физического адреса
- 2) Смещение – берется напрямую – вторая часть физического адреса.

По новому адресу осуществляется поиск уже в физической памяти и доступ к данным

# Рис. Страничная организация памяти



Существуют 2 процесса 0 и 1, у 0 есть 2 страницы у 1 процесса 4 страницы. И Мы видим, что они могут отображаться как угодно, вплоть до того, что 2 виртуальные страницы могут отображаться на одной физической. Зачастую это бывает полезно, простор для манипуляций ограниченный и в этом состоит вся прелесть виртуального адресного пространства.

# Страничная организация памяти. Пример

---

-32битная разрядность адресов

-Размер страницы 4096байт

-VPN длиной 20бит, смещение 12бит (20+12=32бита)

-Преобразуем виртуальный адрес **0\* 43456 323**

-

**№ вирт. страницы    смещение внутри страницы**

-VPN=43456 смещение=323

-Допустим, что в ячейке таблицы страниц по индексу 0\*43456 находится значение фрейма PFN= 0\*1002. Получаем физический адрес 0\*01002323



# таблица страниц (PTE)

---

Если есть таблица страниц, которая содержит преобразование адреса, то необходимо воспользоваться и нагрузить ее дополнительными функциями:

- Добавить защиту доступа
- Добавить дополнительную вспомогательную информацию (например, используется эта вирт.страница или нет, был ли к ней когда-либо осуществлен доступ, была ли в нее осуществлена запись...)

Таблица страниц превращается в сложную структуру данных, которые начинают использоваться множеством всяких дополнительных полей.

Все это нужно, чтобы сохранить память для других процессов, делать все быстро и не плодить десятки новых таблиц с данными – все по возможности хранить внутри таблицы PTE.

# таблица страниц (PTE)

---

Если мы посмотрим на запись в PTE, то мы увидим, что туда можно поместить.

V	R	M	P	PFM
---	---	---	---	-----

V - может ли использоваться данная запись PTE (valid or not) – бит валидности, бит присутствия

R - был ли доступ к этой странице

M – была ли страница модифицирована

P – какие операции разрешены (битовая маска операций)

PFN – номер фрейма (как основной, как осн.нагрузка)

# Преимущества страничной памяти

---

- Легко выделять физическую память
  - Списки свободных фреймов, выделить фрейм – просто удалить из списка свободных
  - Внешняя фрагментация не проблема, т.к. фреймы одного размера
- Естественный подход
  - Всей программе не нужно быть резидентной – это «побочный» продукт
  - Все страницы одного размера
  - Основа – устранение внешней фрагментации

# Недостатки страничной памяти

---

- Внутренняя фрагментация
  - Процессам может быть нужны размеры, не кратные размеру страницы
  - По сравнению с размером адресного пространства, размер страницы очень мал.
- Накладные расходы при обращении к памяти – вначале к таблице страниц, а затем уже к памяти.

**Решение:** аппаратный КЭШ для обращений к таблице страниц (**TLB** translation lookaside buffer – буфер внутри процессора)

# Недостатки страничной памяти

---

- Большой объем памяти, требуемый для хранения таблиц страниц
- 1 PTE на 1 страницу в виртуальном адресном пространстве

Пример: Архитектура x86

- 32-битное АП с 4КБ страницами =  $2^{20}$  PTE = 1048576 записей PTE
- 4 байта на PTE = 4Мб памяти на таблицу страниц
- В Ос создаются отдельные таблицы страниц для каждого процесса, итого, например 25 процессов \* 4Мб = 100Мб
- Соответственно нужны отдельные таблицы страниц для каждого процесса.

**Решение: хранить таблицы страниц в страничной памяти)**

# Страничная организация памяти (обобщение)

---

- Решает разные проблемы, типа фрагментации
- Адресное пространство – линейный массив байтов
- Разделяется на страницы одинакового размера (например 4Кб)
- Использует таблицы страниц для отображения виртуальной страницы на физический фрейм

# Сегментация

---

- Разделяет адресное пространство на логические блоки (стек, код, куча...)
- Виртуальный адрес в виде – сегмент + смещение

Страничная организация предполагает куски памяти, как в примере выше, 4096байт (4Кб), таких кусков можно брать сколько угодно.

Сегментация подходит к распределению памяти более «продвинуто» - есть разные логические сегменты памяти: стек, код программы - имеют свой размер, расположение и права доступа, куча – для динамической памяти программы – все это **отдельные логические блоки памяти**. Их не нужно мешать в одно, поэтому следующим этапом было разделение всего названного и ввод виртуального адреса в виде пары: **Сегмент, смещение**

# Сегментация. Зачем?

---

- Позволяет разделить разные участки памяти в соответствии с их назначением
- Динамический (изменяемый) размер у сегментов

## **2 варианта**

- 1 сегмент на процесс – переменный раздел
- Много сегментов на процесс - сегментация



# Аппаратная поддержка сегментации

---

- Одна пара база-предел(лимит) на сегмент
- Сегменты идентифицируются №, который является индексом в таблице
- Виртуальный адрес = пара <СЕГМЕНТ, СМЕЩЕНИЕ>
- Физический адрес = база сегмента + смещение

## **Недостатки (-)**

- Все недостатки, присущие организации памяти разделами переменной длины присущи и сегментной организации
- Внешняя фрагментация

# Аппаратная поддержка сегментации

---

- Лучшим, как можно заметить является организация этих подходов в один. Давайте объединим страничную и сегментную адресацию
- Архитектура x86 поддерживает и страничную и сегментную адресацию
- Сегменты используются для управления логическими блоками, обычно сегменты большие (помещают множество страниц)
- Сегменты разбиваются на страницы, у каждого сегмента своя таблица страниц

В современных ОС достаточно ограниченно применяются сегменты, много их не применяется.

## **Пример ОС LUNIX**

- 1 сегмент кода ядра, 1 сегмент данных ядра
- 1 сегмент кода пользователя, 1 сегмент данных пользователя
- Все сегменты организованы странично.

# Страничная виртуальная память

---

- Мы предполагали ранее, что вся память резидентная (не рассматривался вариант, что какой-либо страницы может не быть в физ.памяти).
- Сейчас, допустим, что адресное пространство может и не быть полностью резидентным. На практике так в основном и есть – память никогда не резидентна. Процессов сотни, на них выделяется много вирт.памяти и ее не хватит на отображение в физ. Памяти.
- Например 100 процессов  $100\text{П} * 4\text{Гб} = 400\text{ГБ}$  ОП.
- Соответственно какая то часть адресного пространства находится в физ.памяти, какая то часть в некоторой вторичной памяти (понимается дисковая подсистема), абстрагируется от физ. реализации.

# Страничная виртуальная память

---

- С точки зрения ОС важно, что ОС использует основную память, как КЭШ. У ОС есть медленная память (это дисковая подсистема), она может туда загружать процессы и выгружать их оттуда, а основная память используется как ограниченное средство кэширования.
- Принцип работы достаточно простой – нужная страница перемещается в свободный фрейм физической памяти из вторичной памяти.
- А если свободных фреймов нет, то какая-либо страница выгружается на диск, т.е. освобождается драгоценный фрейм физ. Памяти.
- Важно отметить, запись во вторичной памяти происходит только тогда, когда страница в основной памяти была модифицирована, если она не была модифицирована, то соответственно и выгружать нечего.
- Весь процесс происходит прозрачно для программы. Никакая программа не управляет напрямую, какую страницу ей выгрузить в основную память, какую загрузить.
- Менеджер памяти ОС все абстрагирует и делает все прозрачным для программиста, чтобы он не «напрягался» на данную тему, это сложно реализовывать самостоятельно.

# Page fault – Страничное прерывание

---

- Процесс обращается к виртуальному адресу на выгруженной (или загруженной) странице, т.е. страница в принципе отсутствует, при этом обращении и происходит страничное прерывание.

Как этот процесс весь работает:

- Когда страница выгружается, ОС устанавливает **бит присутствия** (`valid` – является ли валидной ячейка) `PTE=0`. И там же в `PTE` записывает куда она была соответственно выгружена.
- Когда же процесс обращается к этой странице, то происходит **исключение**, т.к. `Valid=0`, т.е. бит валидности установлен в 0 – страница не использовалась.
- После того, как произошло исключение ОС передает управление **обработчику страничного прерывания**
- Обработчик находит то **место**, куда была выгружена страница
- Считывает эту страницу в фрейм физической памяти, обновляет `PTE`, ставит бит валидности (присутствия) в 1.
- В физической памяти появляется новая страница.

Процесс достаточно простой, если страницы нет, то сама же ОС ее загрузит обратно, или выгрузит, когда ей это надо.

# Загрузка по требованию

---

Еще один ключевой механизм работы менеджера памяти. Практически все страницы памяти загружаются по требованию.

Смысл: страницы загружаются в основную память только тогда, когда к ним происходит **непосредственное обращение**.

Предсказать заранее, какая страница потребуется в будущем – сложно (это как гадать на кофейной гуще).

Сегодня для этого разработано множество алгоритмов, есть разные подходы.

Самый логичный из них - **кластеризация страниц**.

ОС ведет учет страниц, которые обычно загружаются вместе, даже если они расположены в различных местах виртуальной памяти или физической памяти.

Если идет обращение к одной из них, то ОС **загружает все страницы их этого кластера**. Это естественный подход к тому, как можно провести исходную оптимизацию этого алгоритма.

Можно даже предоставить программисту возможность определять такие кластеры.

# Механизм замещения страниц

---

Допустим у нас заняты все фреймы(страницы) физической памяти, а нам нужно еще загрузить одну страницу.

? – какую из страниц физ. памяти выгрузить? Ведь страниц много и это выбора многое зависит.

**Алгоритм замещения страниц** простой:

1. Выбрать страницу, которую не понадобится в ближайшем будущем
2. Выбрать страницу, которая не была модифицирована, чтобы обойтись без записи ее на диск.
3. Чтобы обойтись без замены в ОС есть пул свободных страниц.

Но проблема замещения страниц существует.

Посмотрим последовательность действий в ОС, когда у нас загружается программа.

# Как загружается программа?

---

1. Для создания нового процесса создается новый PCB (процесс контрол.блок).
  2. Создается таблица страниц для этого процесса, которая описывает его виртуальное адресное пространство
  3. Образ программы на диск размещается блоками в адресное пространство (он не копируется, а неким образом размещается)
  4. Строится таблица страниц, указатель на которую уже есть в PCB  
Все PTE имеют бит присутствия =0 (виртуальное адресное пространство в начале процесса полностью пустое, в плане отображения на физическую память – ни одна страница не присутствует)  
В эти же PTE заносится информация о нахождении этих физических страниц уже во вторичной памяти, т.е. на диске
  5. Передает управление на точку входа этого процесса – исполнение первой же инструкции приводит к страничному прерыванию (page fault)
  6. Обработчик прерывания загружает эти страницы из дисковой памяти
- Т.О. никто, ничего заранее не подгружает, все загружается через «page fault» по требованию. Именно так все работает.



# Механизм замещения страниц

---

Принцип локальности определяется 2 подходами:

- Одна загрузка –много обращений (**локальность по времени**)
- Обращение к страницам рядом с уже загруженной (**локальность по расположению**).

**Загрузка страниц** по требованию может быть частой, а может и не такой частой, **зависит от:**

- Локальности
- Политики замещения страниц
- Объема физической памяти
- «рабочего множества», так называемого

Смысл алгоритма замещения страниц – уменьшить число страничных прерываний (page fault) путем выбора **лучшей страницы** для выгрузки.

Лучшая – та, к которой **вообще больше не будет обращений**, т.е. освобождает место в памяти и решаем все вопросы, все будет работать быстро.

Далее рассмотрим разные алгоритмы, которые были предложены к решению данной проблемы (фундаментальной и основополагающей, как оказалось). Это как в алгоритмах планирования, они существенно влияют на производительность, отзывчивость системы. В менеджере памяти тоже есть ниша, которая оказывает большое влияние.