Операционные системы

Курс лекций для гр. 4057/2

Лекция №7

Вопросы к лекции 6

1. Напишите псевдокод производителя-потребителя с ограниченным буфером, используя примитивы send(proc_name, &m) и receive(proc_name, &m), где &m – указатель на место в памяти, где может находиться элемент (порция) данных (или что-либо еще).
2. (У.Столлингс, 2002) Покажите, что сообщения и семафоры обладают эквивалентной функциональностью, для чего реализуйте следующее:

   А. Передачу сообщений с использованием семафоров. *Указание*: сделайте это с помощью буфера для хранения почтовых ящиков, каждый из которых представляет собой массив сообщений.

   Б. Семафоры с использованием передачи сообщений. *Указание*: добавьте в систему синхронизирующий процесс.
3. Почему синхронная схема коммуникации считается более надежной, чем асинхронная?
4. Выведите формулу закона Амдала.
5. Всегда ли логически независимые процессы (т.е., взаимодействие которых ограничивается только конкуренцией за ресурсы) имеет смысл выполнять на разных процессорах? Если не всегда, то когда экономически более эффективно их выполнение на одном и том же процессоре?
6. Если у двух взаимодействующих процессов:

1) преобладают отношения конкуренции за доступ к критическим ресурсам, то... (а или б?)

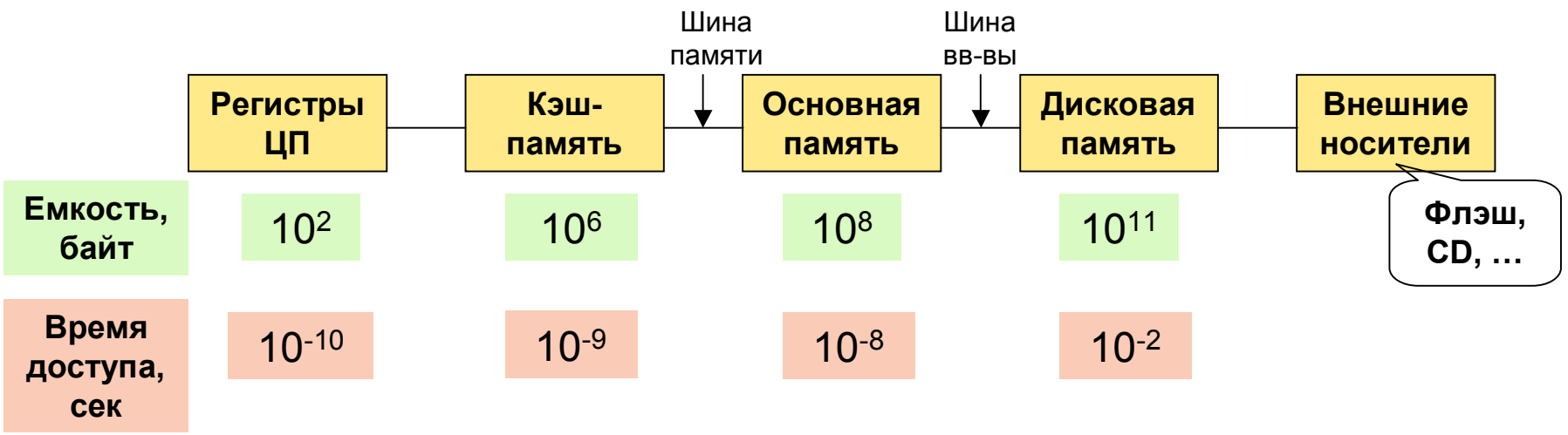
2) преобладают отношения кооперации (сотрудничества), то... (а или б?)

   а) их нужно выполнять на одном и том же процессоре

   б) их нужно выполнять на разных процессорах

Содержание

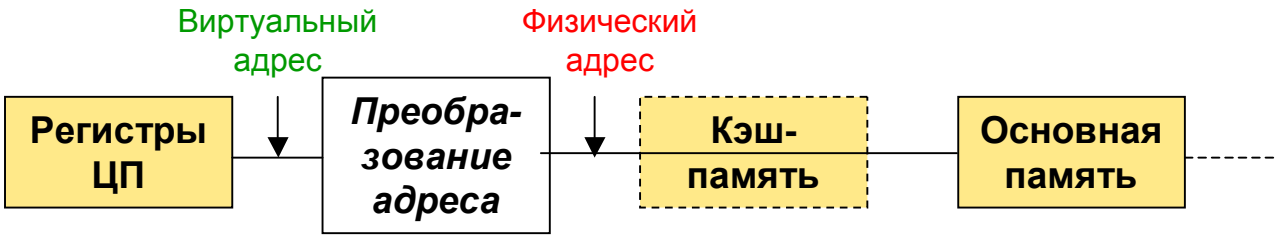
Иерархия памяти



Чем больше емкость памяти, тем больше время доступа до заданной единицы данных

- Разрывы в быстродействии:
  - процессора и основной памяти – 2 порядка величины
  - основной и дисковой памяти – 6 порядков!
- Быстродействие основной памяти растет гораздо медленнее быстродействия ЦП, а у дисковой памяти не растет совсем

Виртуальные и физические адреса



Виртуальный (логический) адрес - адрес в процессоре во время выполнения программы: в счетчике команд, регистре адреса или в самой команде

Физический адрес - адрес (номер) ячейки основной памяти

Основная задача управления памятью - это отображение *виртуальных адресных пространств* процессов на *физическое адресное пространство* оперативной памяти: ВАП $\rightarrow$ ФАП

Размер ВАП определяется архитектурой процессора:

2^{16}= 64 K адресов при 16-битовой адресации (тогда обычно ВАП < ФАП)

2^{32}= 4 G адресов при 32-битовой (обычно ВАП >ФАП)

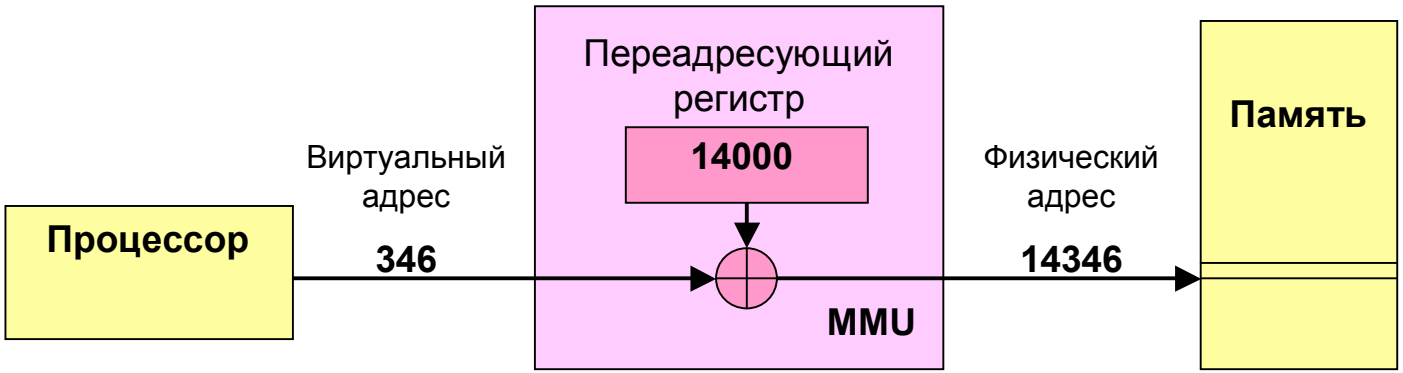
Схема подготовки и выполнения модульных программ

Переадресующий регистр

Динамическое отображение адресов может выполняться:

– программно: перемещающим загрузчиком – это медленно

– аппаратно: диспетчером памяти – MMU (memory management unit); в простейшей форме это *переадресующий (relocation) регистр:*

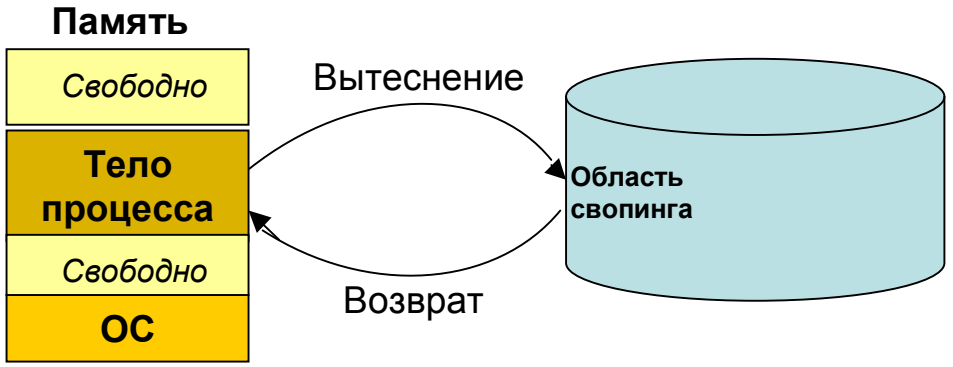


Такая схема применялась для расширения ВАП в ранних, 8 и 16-разрядных архитектурах, где ВАП < ФАП (PDP-11, Intel 80286)

Напр., MS-DOS добавляла 4 или 6 разрядов к 16-разрядному адресу процессора, увеличивая адресуемую память с 64 Кб до 1 или 4 Мб . На этом же были основаны программные *расширители памяти* MS-DOS (FarLap, DOS4GW и др.)

Борьба с нехваткой памяти (1)

В мультипрограммной ОС ограниченная емкость памяти может преодолеваться с помощью *свопинга* (swapping – обмен):

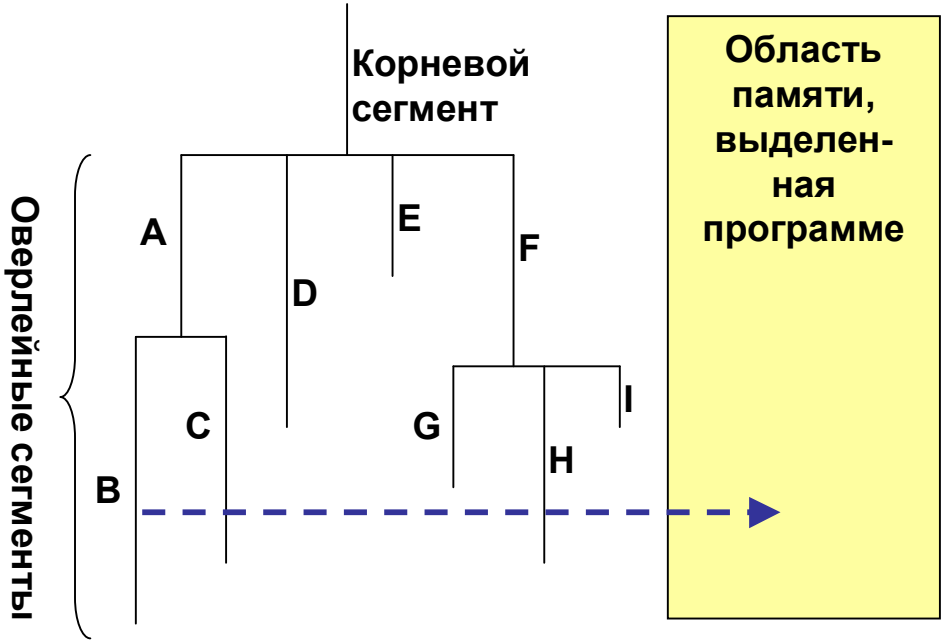


Тело ждущего или готового процесса вытесняется (копируется) временно на диск, если свободной памяти не хватает для очередного выполняемого процесса (диспетчерское состояние: приостановка)

Ранние ОС (напр., Unix для PDP-11, Windows 3.1) широко использовали свопинг для мультипрограммирования при ограниченной емкости памяти

Борьба с нехваткой памяти (2)

Слишком длинная программа – преобразуется в модуль с оверлейной (overlay) структурой (т.е., с запланированным перекрытием)



- Каждый сегмент – одна или несколько подпрограмм и/или сегментов данных
- Корневой сегмент постоянно присутствует в памяти, а оверлейные сегменты загружаются по мере необходимости
- Сумма длин ветвей может значительно превышать размер доступной области памяти
- Здесь статическое связывание и динамическое отображение адресов

Оверлейная структура создается сборщиком по описанию дерева программы и поддерживается оверлейным загрузчиком

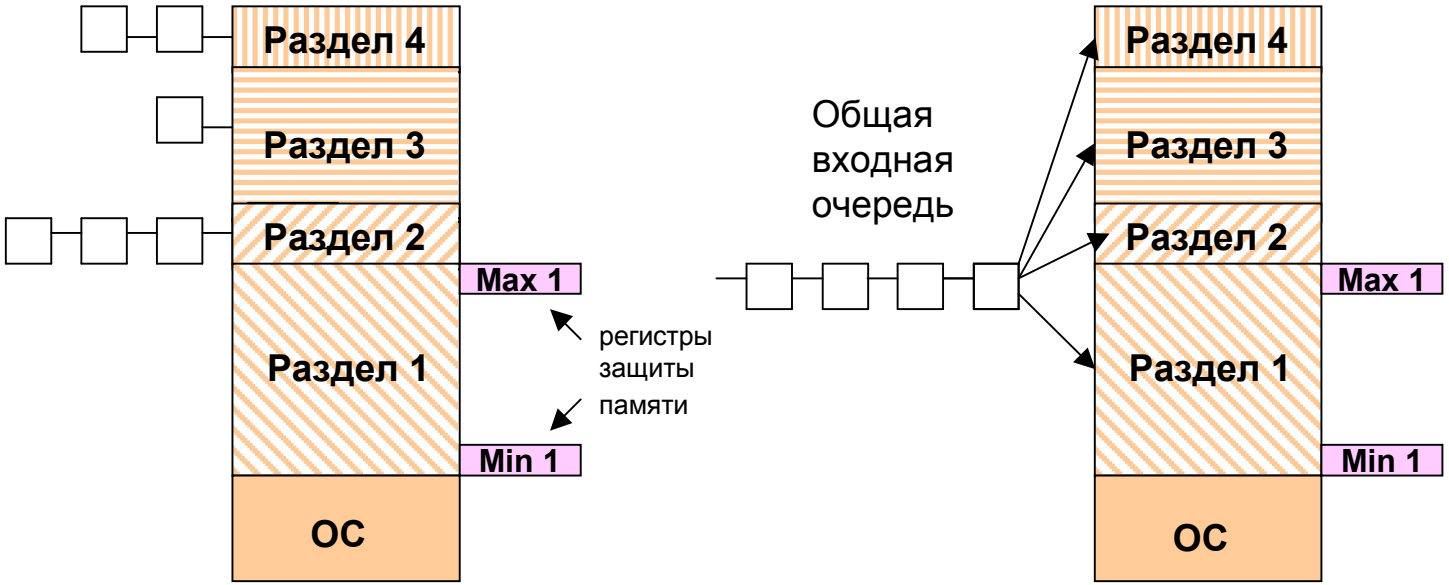
Связное распределение памяти

Процессу предоставляется одна *непрерывная* область памяти:

– в однопрограммном режиме – вся свободная память

– в МП режиме каждому процессу представляется *раздел памяти;* число разделов = коэффициенту МП

1. Фиксированные разделы памяти (MFT)

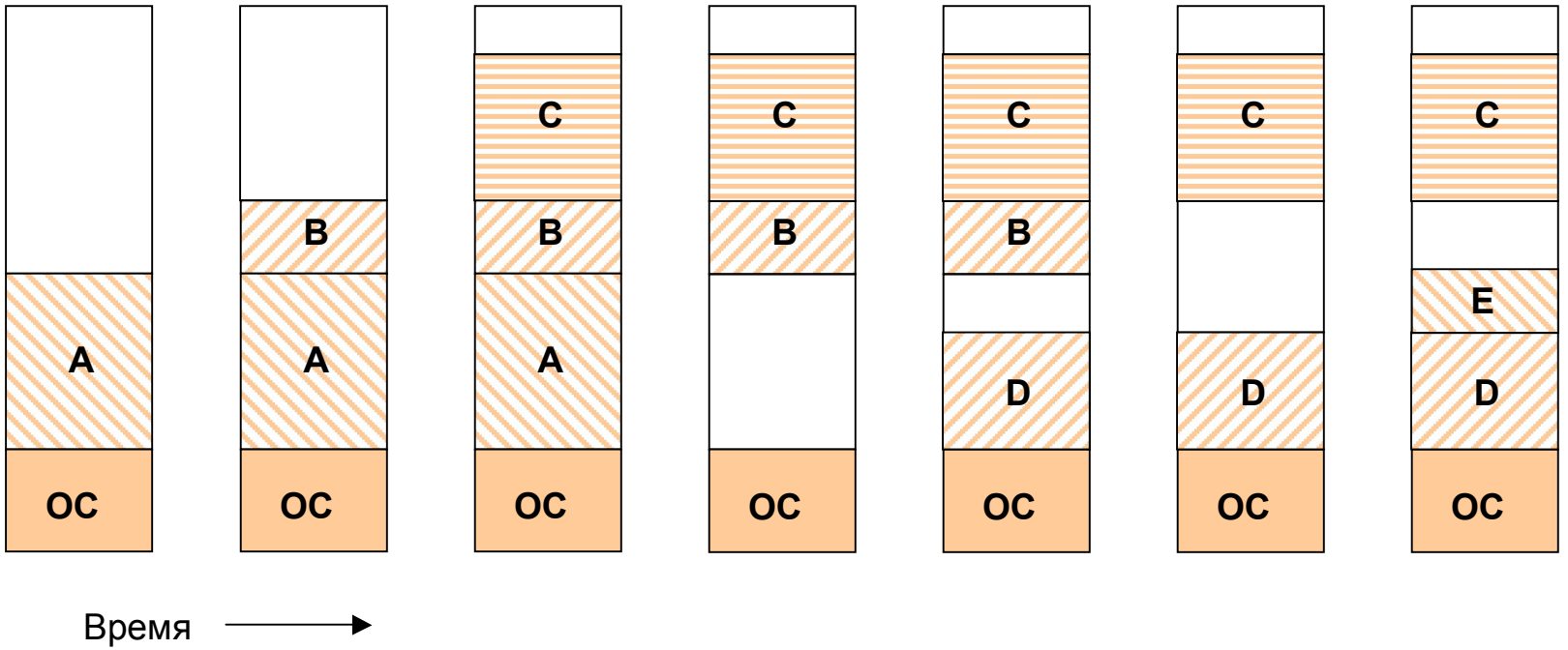


Пара регистров защиты памяти - регистров max и min адреса раздела – в составе контекста процесса; выход адреса за их пределы приводит к прерыванию по защите памяти

Связное распределение памяти (2)

2. Переменные разделы памяти (MVT)

Долгосрочный планировщик определяет число и размеры разделов, исходя из текущей очереди заданий и решает задачу статического распределения памяти



Распределение свободной памяти

В какую из свободных областей («дыр») загружать очередной процесс из общей входной очереди?

- First-fit: в первую подходящую дыру из списка дыр
- Best-fit: в наименьшую подходящую дыру
- Worst-fit: в наибольшую дыру

Все три подхода дают приблизительно одинаковые результаты в смысле возможно более «плотного» заполнения памяти

Проблема фрагментации памяти

- наличия в памяти неиспользуемых областей

- *Внешняя фрагментация* – дыры между разделами, возникающие по двум причинам:
  - в освободившийся раздел загружается процесс меньшего размера, чем завершившийся
  - освободившийся раздел недостаточен по размеру ни для одного процесса, ожидающего освобождения памяти
- *Внутренняя фрагментация* – дыры в конце разделов из-за *гранулярности* – дискретности размера выделяемой памяти порциями в 1 блок

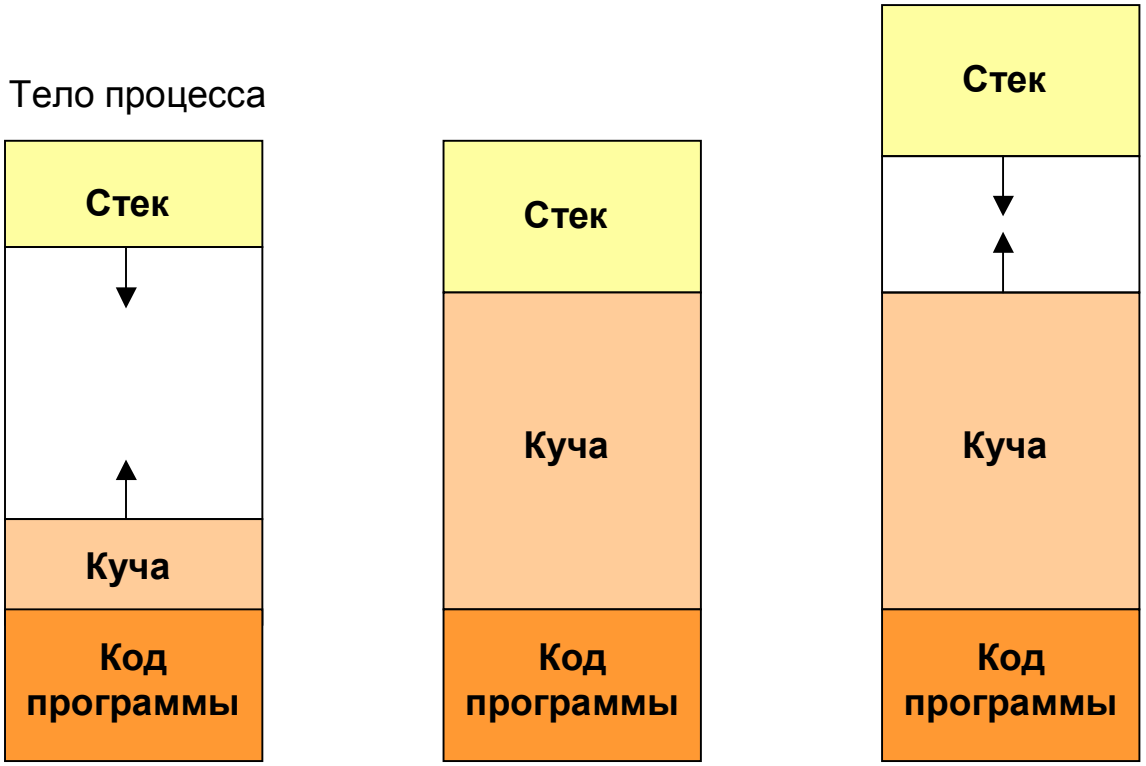
Борьба с внешней фрагментацией – *уплотнение:* слияние дыр путем перемещения тел процессов

- адреса в командах программы пересчитываются, как при загрузке

Уплотнение дорого и медленно: выполнение всех программ приостанавливается на время уплотнения – иногда на секунды

Проблема динамического изменения размера разделов

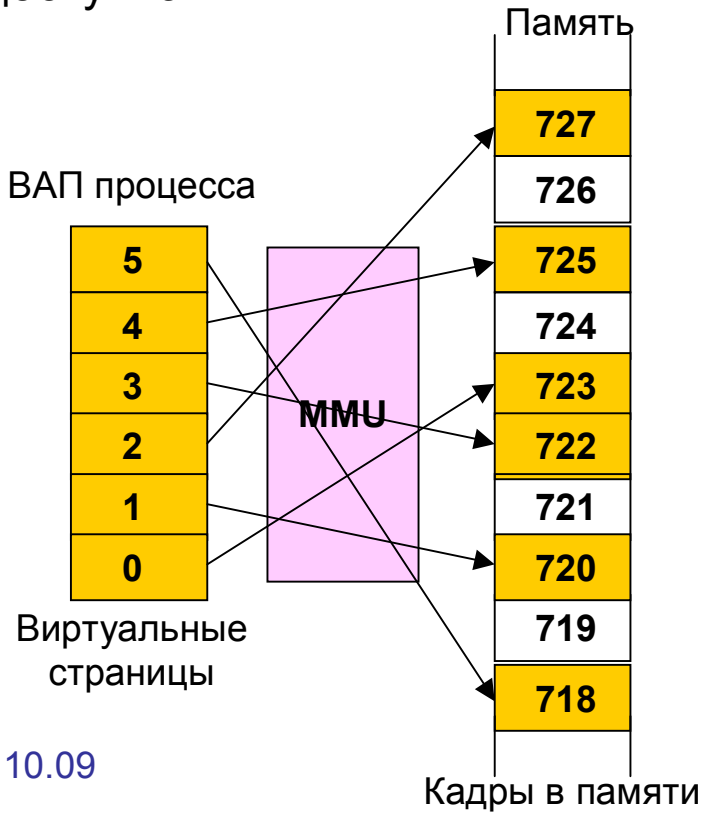
- Увеличение потребности памяти в ходе выполнения процесса, когда переполняется куча или стек, или он динамически подгружает DLL-программу

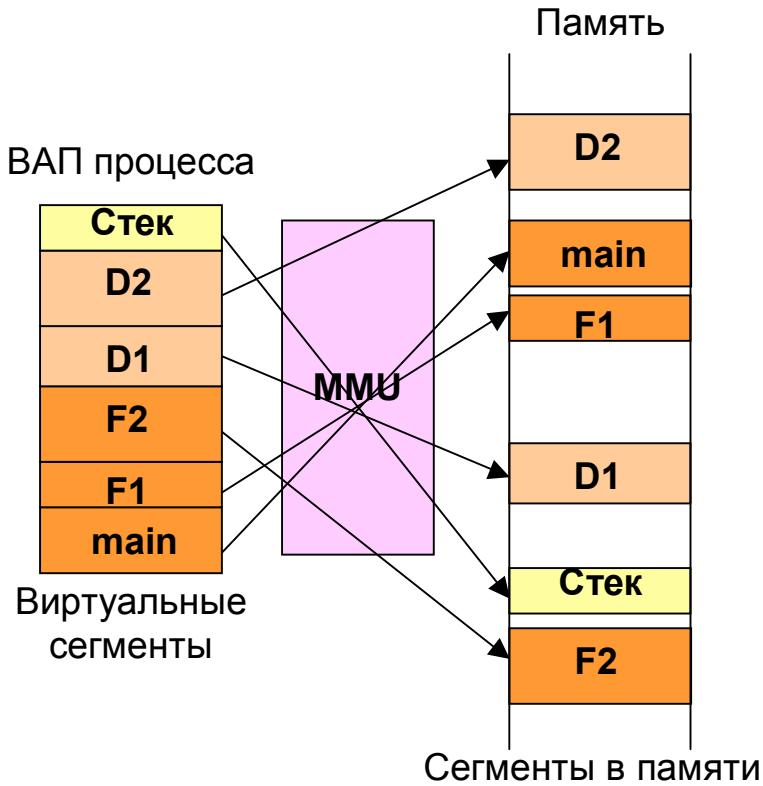


Динамическое увеличение размера раздела может потребовать перемещения других процессов, как и при уплотнении

Несвязное распределение памяти

- Тело процесса режется на куски (*страницы* или *сегменты*) в ВАП
- Страницы - это куски одинакового размера, сегменты – логические части программы: коды (головная программа, функции), стек, данные
- Виртуальные страницы или сегменты отображаются на физические - несвязные области в ФАП - с помощью диспетчера памяти (MMU)
- Физические страницы или сегменты могут занимать память везде, где она доступна





Несвязное распределение памяти vs. связное

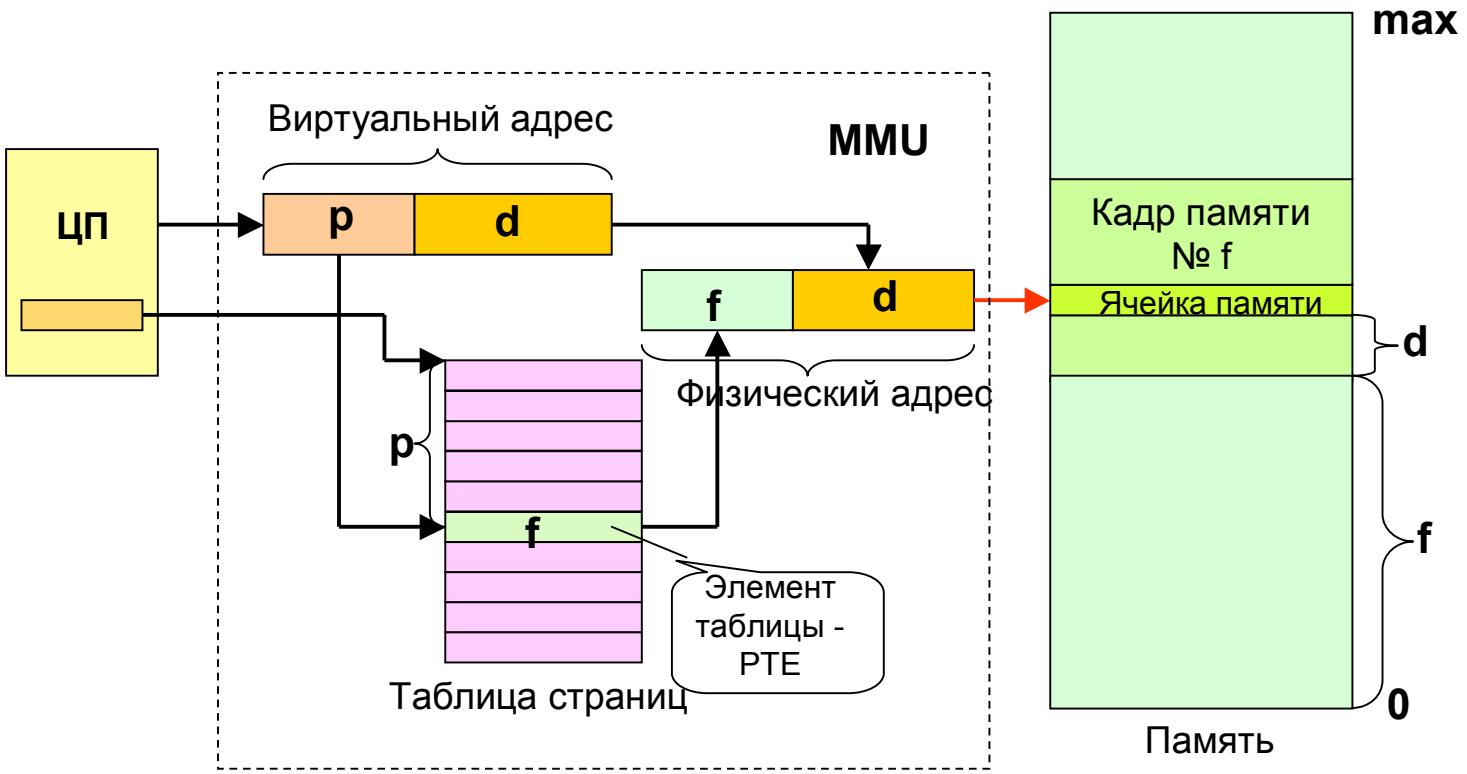
Преимущества

- Проще выделение свободной памяти новому процессу: несмежными кусками
- Проще перемещение частей тела процесса в памяти: достаточно изменить базовый адрес страницы или сегмента в диспетчере памяти
- Снимаются проблемы внешней фрагментации:
  - она отсутствует при страничном распределении (с точностью до размера блока выделяемой памяти)
  - перемещение сегментов для уплотнения – быстрое и не требует остановки вычислительного процесса
- Проще изменение физического адресного пространства процесса:
  - новые кадры могут заниматься в любом свободном месте памяти
  - сегменты могут расти, не мешая друг другу
- Упрощается совместное использование и дифференцированная защита (читать, писать, выполнять) частей программы

Издержки

- Увеличивается эффективное время обращения к памяти из-за преобразования адреса в MMU
- Увеличивается время переключения контекста – на время перезагрузки адресной информации в MMU
- При страничной организации увеличивается внутренняя фрагментация

Схема аппаратной трансляции адреса при страничной организации памяти



p – номер страницы - индекс входа в таблицу страниц

d – смещение - младшие разряды физического адреса

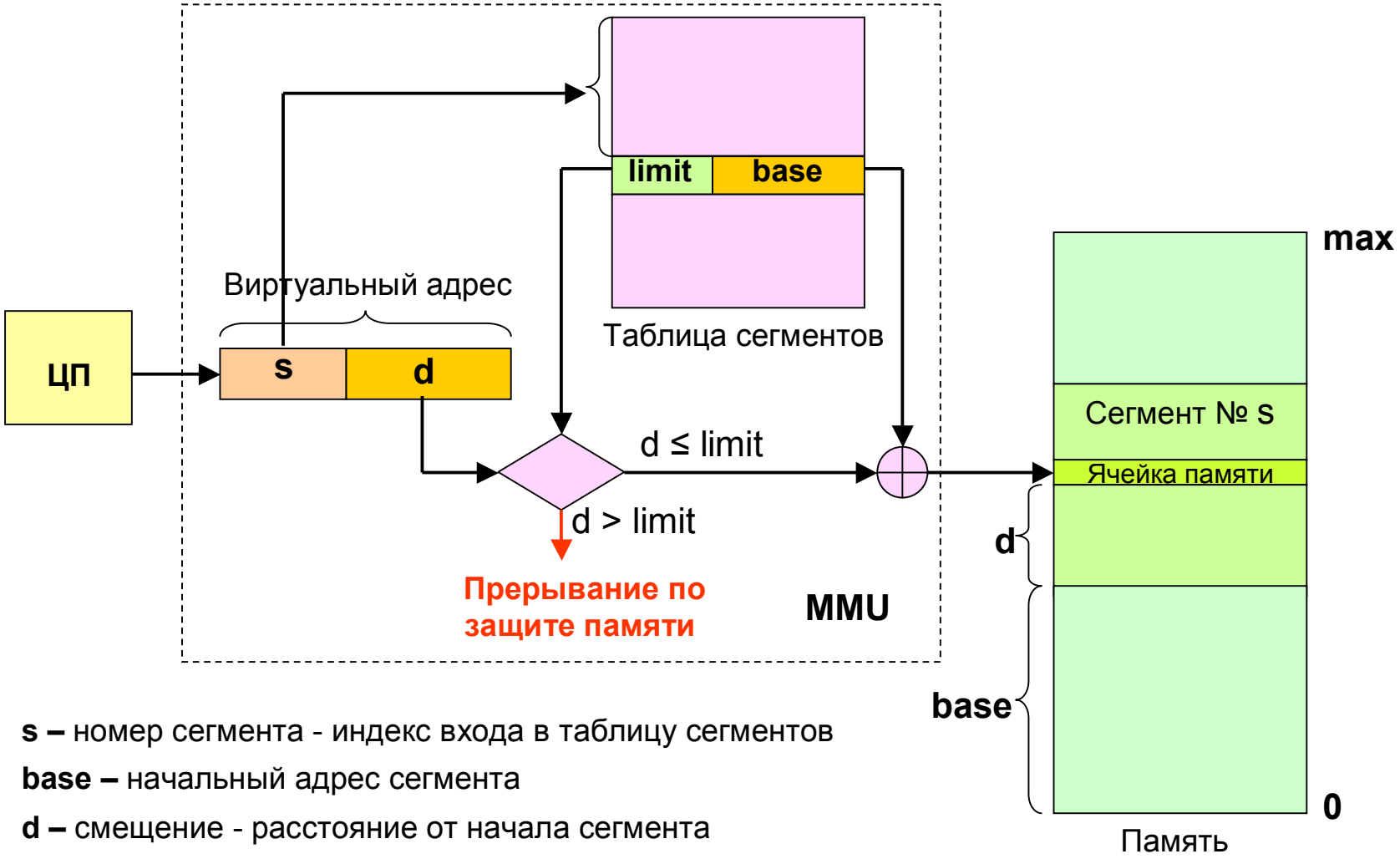
f – номер кадра – старшие разряды физического адреса

Размер страницы в байтах всегда 2^N

Таблицы страниц

- Таблица страниц (ТС) процесса хранится в рабочей области ядра
- Указатель на ТС текущего процесса – в регистре процессора
- У текущего процесса элементы ТС хранятся в регистрах ЦП, в кэш-памяти или основной памяти
- Специализированная кэш-память для элементов ТС называется Translation Look-aside Buffer (TLB)
  - Доля случаев, когда нужный элемент найден в TLB, называется *вероятностью попадания (hit ratio)*
  - При N - числе входов в TLB - от 16 до 512, вероятность попадания - в диапазоне от 80% до 99%
  - В Intel 80486 (1992 г.) N=32; в новейших процессорах N=4 К и более
- Сразу после переключения контекста процессов эффективное время доступа к памяти может возрасти из-за отсутствия нужных страниц в TLB

Схема аппаратной трансляции адреса при сегментной организации памяти



s – номер сегмента - индекс входа в таблицу сегментов

base – начальный адрес сегмента

d – смещение - расстояние от начала сегмента

limit – длина сегмента – максимальный размер смещения

Сегментное распределение памяти vs. страничное

Преимущества

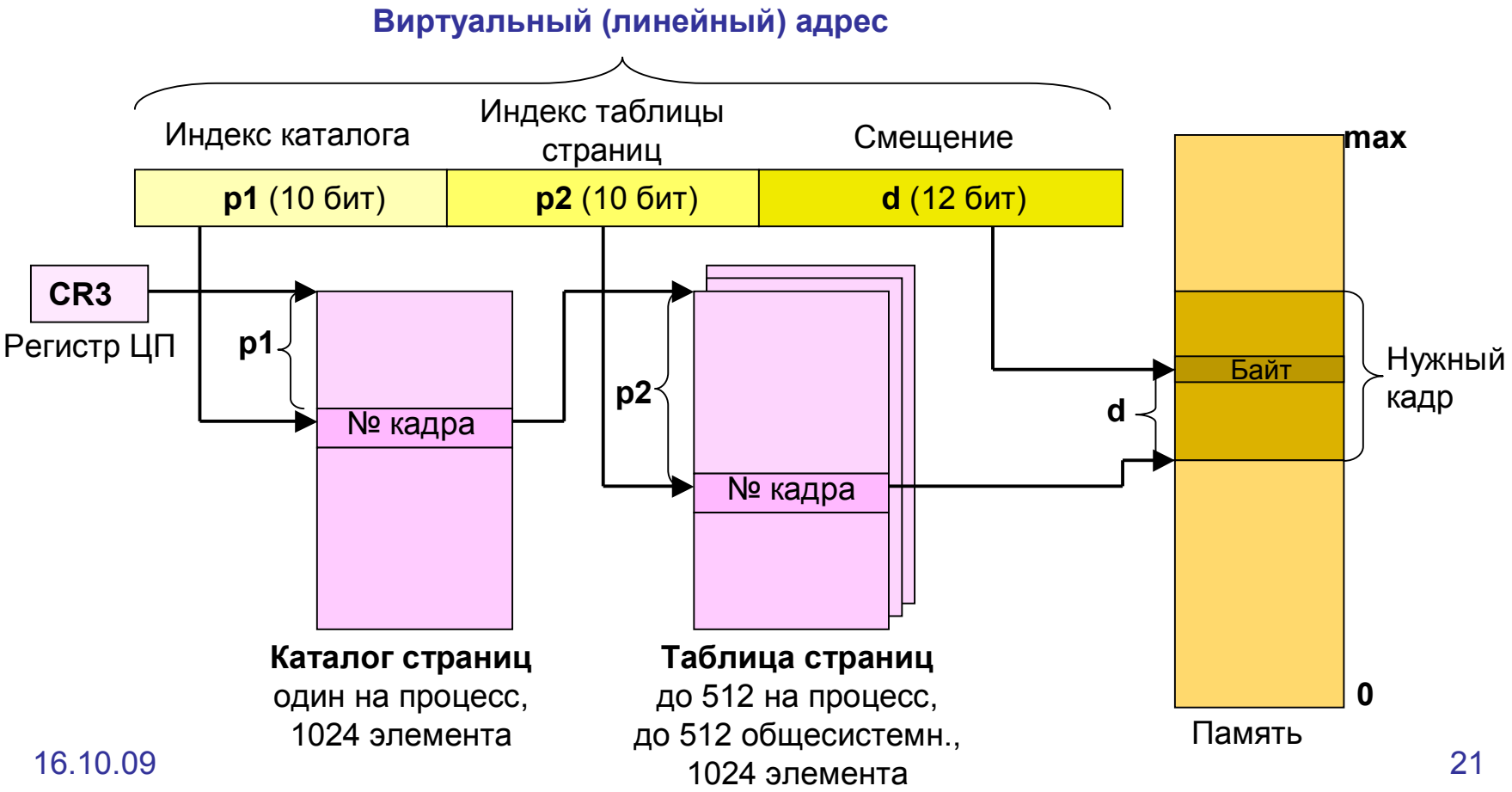
- Проще защита и разделение кода и данных
- Проще динамическое выделение памяти под стек и кучу
- Меньше внутренняя фрагментация
- Меньше размер таблиц сегментов

Недостатки

- Внешняя фрагментация (такая же, как при распределении разделами, однако уплотнение гораздо быстрее)
- Сложение выполняется дольше конкатенации -> больше время доступа к памяти (эта разница невелика в современных процессорах)

Многоуровневая страничная память

- способ борьбы со слишком большим размером таблиц страниц для больших программ. Пример: схема трансляции двухуровневых страниц в процессоре Intel:



Многоуровневая и смешанная организация памяти

Другие примеры многоуровневой страничной памяти:

- **IBM/370: 24-битовый виртуальный адрес: p1= 8 бит, p2 = 4, d = 12**
- **VAX/VMS: p1= 2 бит, p2 = 21, d = 9**
- **SPARC: трехуровневая страничная организация при 32-битовом адресе**
- **Motorola 68030: 4-уровневая страничная организация при 32-битовом адресе**
- **64-битовый адрес в SPARC 10: p1= 32 бит, p2 = 10, p3 = 10, d = 9**
- **Режим PAE (Physical Address Extension) в Pentium Pro, адресующий до 64 Гб: p0=2 бит, p1=9, p2=9, d=12 при 8-байтных элементах таблиц каталогов и страниц**

В типичной для современных процессорных архитектур смешанной - сегментно-страничной схеме объединяются достоинства двух способов

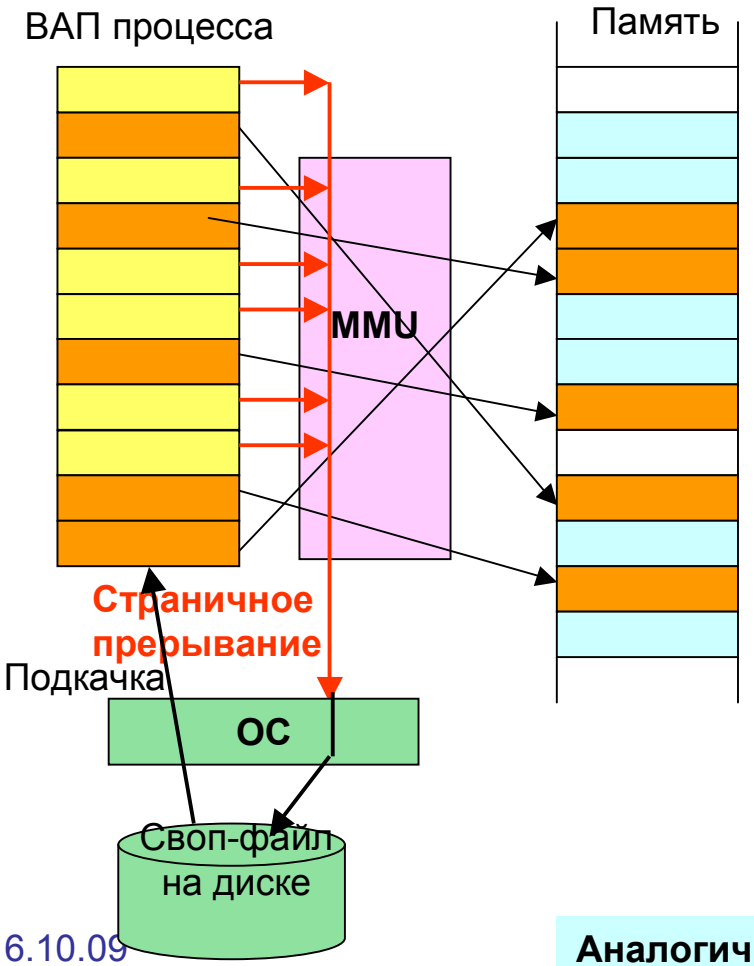
- **Сегменты отображаются на страницы; напр., трехуровневая схема: s, p, d**
- **Четырехуровневая: s, p1, p2, d - в Intel x86**

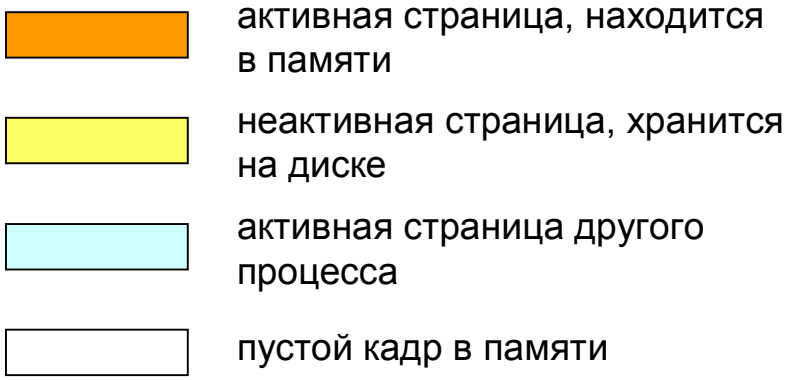
Страничная организация памяти используется в некоторых бездисковых ОС для встроенных систем (напр., Windows CE)

Виртуальная память

Цель: увеличить суммарный размер параллельно выполняемых процессов при ограниченном ФАП

Идея: организовать основную память как кэш для диска





Аппаратная поддержка: страничное прерывание *(page fault)* - прерывание при отсутствии страницы в памяти

Обработка прерывания: подкачка отсутствующей страницы – *страничный обмен (paging, или swapping),* на время которого процесс блокируется

Аналогично – для сегментов

Элемент таблицы страниц – дескриптор страницы

Типичная структура дескриптора активной страницы (так в Intel)

Бит активности **A** - флаг присутствия страницы в памяти

Z – запрет вытеснения (страница – резидентная в памяти)

Флаги **R** (прочитано) и **M** (модифицировано) обновляются аппаратно и могут использоваться программой вытеснения

В состав битов защиты входят флаги R (только читать), RW (читать/писать), E (выполнять код) и др.

В дескрипторе неактивной страницы может быть записан ее адрес на диске

Обработка страничного прерывания:

- Если есть свободная память, загрузить в нее запрошенную страницу
- Если свободной памяти нет, то:
  1) найти страницу, подлежащую вытеснению из памяти (***замещение*** страниц)
  2) сохранить ее на диске, если она содержит данные и помечена как **M** (помеченную как **R** сохранять не нужно)
  3) загрузить требуемую новую страницу и обновить дескрипторы новой и вытесненной страниц (у неактивной страницы структура другая – она содержит адрес страницы на диске)

Алгоритмы замещения страниц

Цель: минимизировать частоту страничных прерываний.

Минимум достигается, если каждый раз вытеснять страницу, которая дольше всего не понадобится – однако эта страница не известна

Базовые стратегии:

1. Предыстория использования активных страниц не учитывается: вытесняется случайная страница. Иногда результаты вполне приемлемые. (NB: этот подход применяется в аппаратной кэш-памяти).
2. Экстраполяция предыстории на будущее
   - 2.1 FIFO: вытесняется наиболее старая страница (т.е., дольше всего находившаяся в памяти)
     - Достоинство - простота реализации (кольцевой список номеров страниц)
     - Недостаток: часто используемая страница вытесняется, чтобы сразу же загрузиться обратно.
   - 2.2 LRU (Least Recently Used): вытесняется дольше всего неиспользуемая страница
     - Недостаток: работает плохо, если процесс циклически обращается к последовательности страниц, которые не помещаются в память одновременно

Аппроксимации LRU (1)

Реализовать точный LRU - слишком дорого! Поэтому применяются различные приближения к алгоритму LRU

NFU (Not Frequently Used): вытесняется редко использовавшаяся страница

- С каждой страницей связан программный счетчик числа обращений к ней Чтобы не обновлять его при каждом обращении к странице, это делается периодически, по таймеру (напр., каждые 20 мс – «тик» процессора)
- При обновлении к значению счетчика прибавляется бит R – отметка чтения страницы в ее дескрипторе, после чего бит R обнуляется (т.е., значение счетчика вычисляется приближенно)
- Для вытеснения выбирается страница с наименьшим значением счетчика

Недостаток алгоритма NFU в том, что он «ничего не забывает»

Аппроксимации LRU (2)

Алгоритм **«Старение»** (Aging): аппаратно поддерживается несколько битов счетчика в дескрипторе страницы (напр., 8 бит)

- Регулярно (напр., каждый «тик») этот байт сдвигается вправо, и в старший бит записывается 0 ("старение" страницы)
- При каждом обращении к странице в старший бит помещается 1
- Вытесняется страница с наименьшим числом в счетчике

Недостатки: как и в NFU, слишком грубый счетчик и дорогое регулярное *программное* обновление счетчиков *всех* активных страниц

Аппроксимации LRU (3)

NUR (Not Used Recently) лучше тем, что нет программного счетчика: он использует *аппаратное* обновление грубого – одно- или двухбитового - счетчика в дескрипторе страницы (если это поддерживается процессором)

- Бит **R** устанавливается в 1 при чтении страницы, бит **M** - при ее изменении
- Периодически бит **R** сбрасывается на 0 (**M** не обнуляется, чтобы помнить, что такую страницу нужно переписывать на диск).
- Коды **R, M** в порядке убывания приоритета для вытеснения:
  - **R**=0, **M**=0 - наилучший кандидат на вытеснение (не было ни обращений, ни записи по крайней мере после последнего сброса **R** в 0)
  - **R**=0, **M**=1 - обращений не было, страница изменена
  - **R**=1, **M**=0 - обращение было, страница не изменена
  - **R**=1, **M**=1 - интенсивно используемая страница

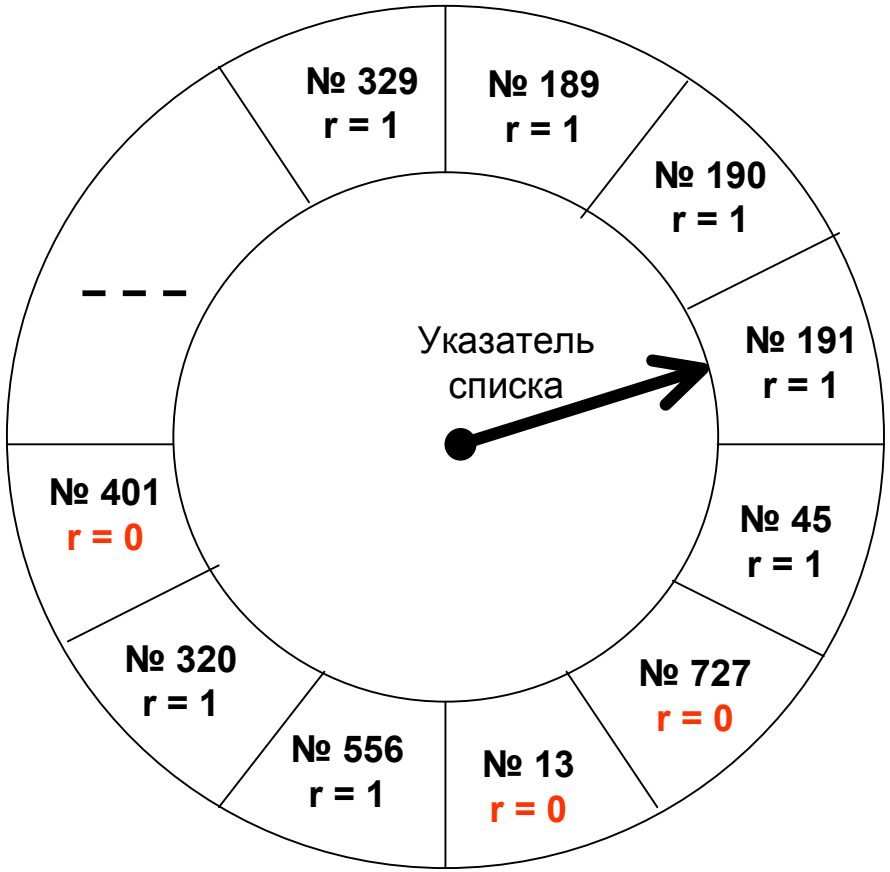
Аппроксимации LRU (4)

Алгоритм второго шанса, или **Часовой (Second Chance,** aka **Clock) -** однобитовый счетчик попыток вытеснения

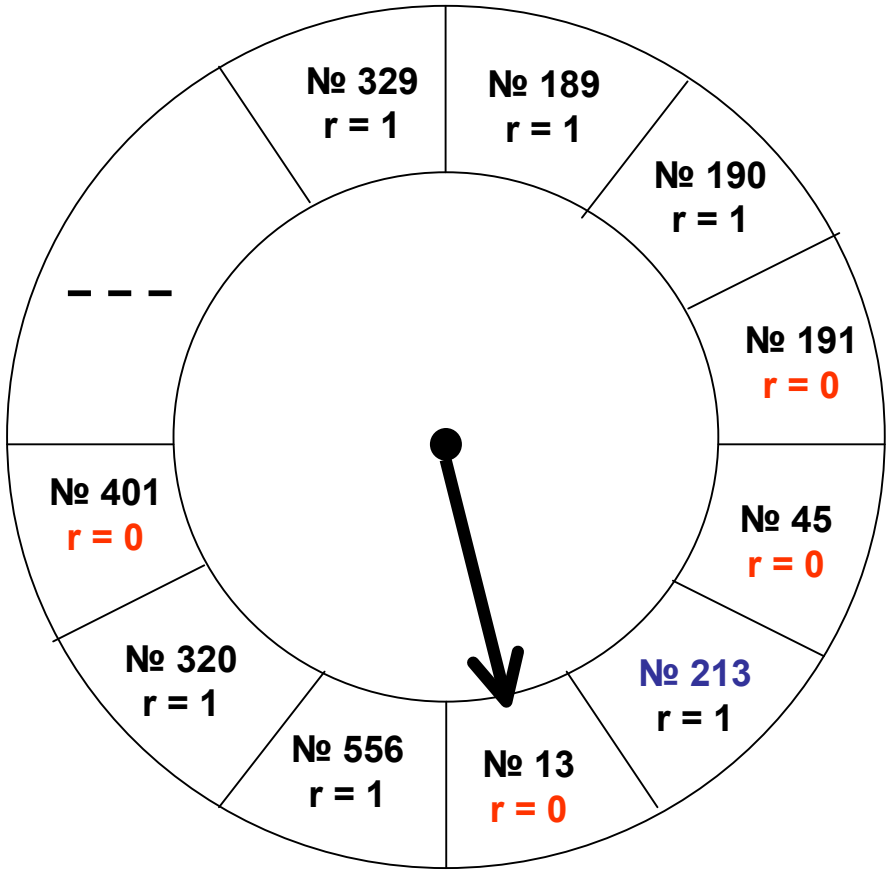
- Кольцевой список номеров активных страниц; с каждым номером связан *ссылочный (reference)* бит **r** – счетчик использования
- При каждом обращении к странице ее **r** устанавливается в 1
- При каждом страничном прерывании ОС проверяет **r** очередной страницы:
  - если **r = 0**, то эта страница замещается новой с **r := 1** и указатель передвигается на следующую страницу
  - иначе **r := 0** (странице дается "второй шанс" остаться в памяти) и проверяется следующая страница

Так страницы разбиваются на два класса: "молодые", недавно использованные (**r** = 1) и "старые", давно неиспользуемые (**r =** 0) – кандидаты на вытеснение

Часовой алгоритм



Запрошена отсутствующая страница № 213



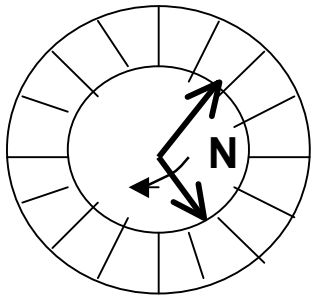
Страница № 213 заместила страницу № 727

Реализация часового алгоритма в одной из последних версий Linux

Еще одна особенность реализации: периодическое (раз в сек.) обновление списков

Варианты часового алгоритма в версиях Unix/Linux

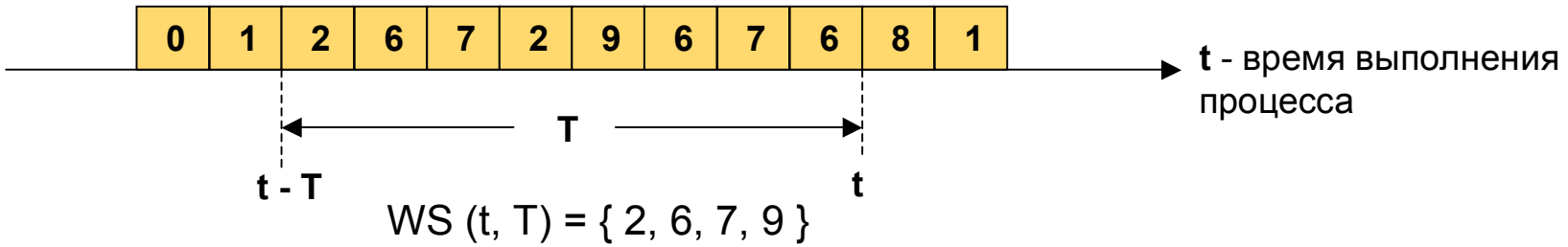
- Двухбитовый счетчик (как в NUR): R и M
  - при M=1 страница не вытесняется
- 8-битовый счетчик:
  - ++ при каждом обращении к странице
  - регулярное уменьшение счетчика у всех страниц (как в алг. “Старение”)
- Две стрелки, вращающиеся синхронно:
  - первая обнуляет счетчик r
  - вторая с отставанием на N страниц выбирает замещаемую страницу
  - эффективно при большом количестве страниц
  - N – параметр настройки



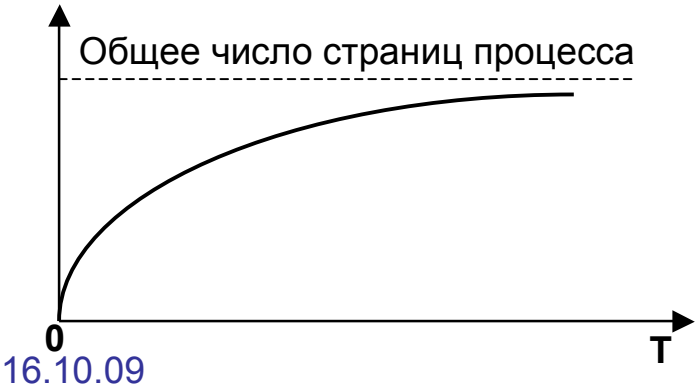
Алгоритм рабочего набора

Уточненный алгоритм NUR: вытесняется страница, к которой не было обращений последнее время (фиксированный отрезок времени)

Рабочий набор (working set, WS) - множество страниц, к которым было обращение за последние T сек. работы процесса; WS = f (t, T)



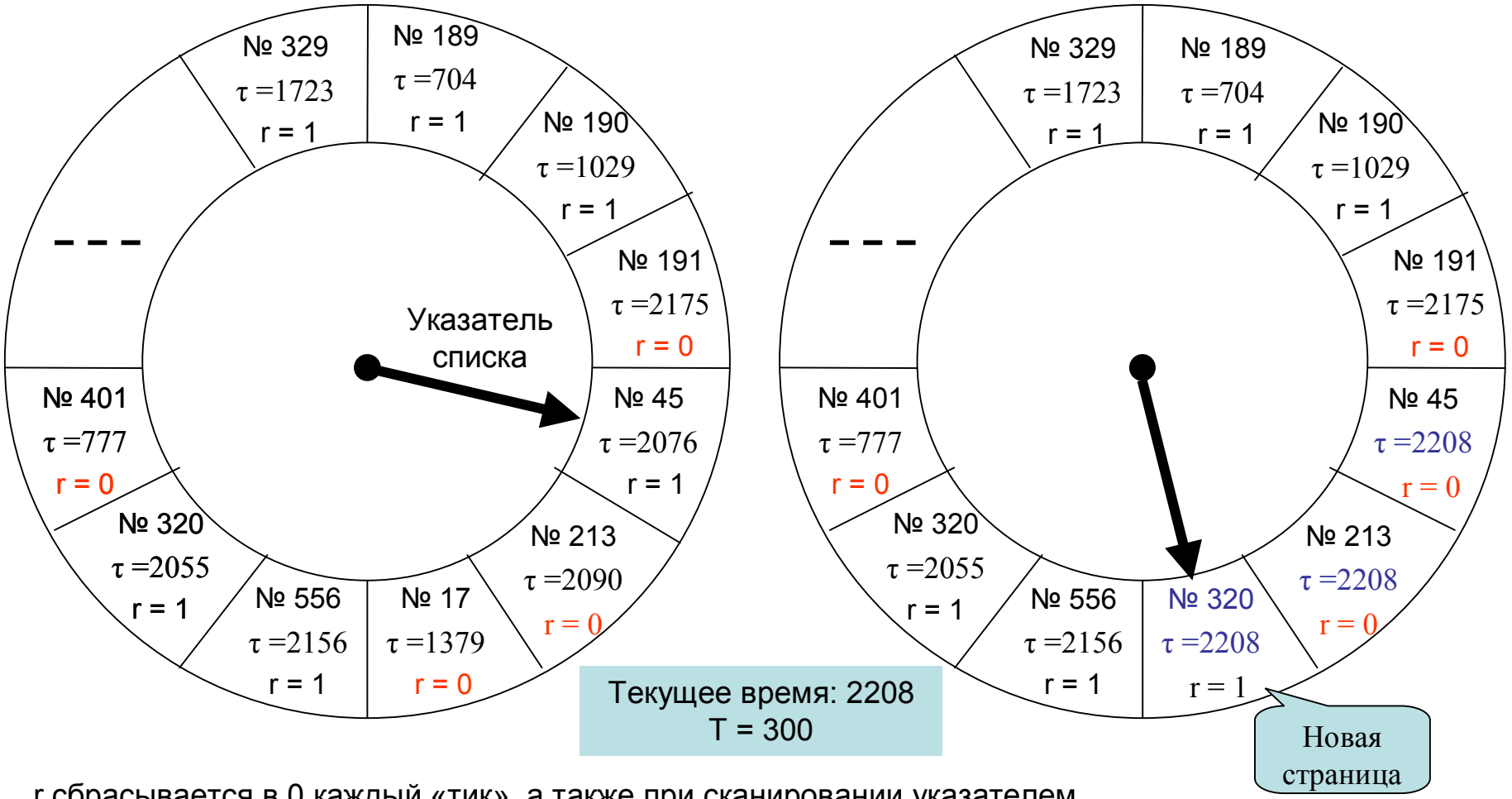
|WS| - число страниц рабочего набора



Идея алгоритма WS: вытеснять активную страницу, не входящую в рабочий набор – одну из тех, к которым не было обращения последние T сек или дольше

Время последнего обращения τ измеряется приближенно – с точностью до «тика» или еще грубее

Алгоритм WSClock



r сбрасывается в 0 каждый «тик», а также при сканировании указателем

Стр. № 213: 2208 – 2090 = 118 < T - страница в рабочем наборе, она сохраняется

Стр. № 17: 2108 – 1379 = 729 > T - страницы нет в рабочем наборе, она вытесняется

Приемы улучшения алгоритмов (1)

Общая идея – буферизация (“сглаживание рывков")

1. Буферизация с помощью "отложенных на время" действий

а) Постоянно поддерживаемый буфер свободных кадров: он позволяет не ждать записи на диск старой модифицированной страницы, а сразу же загрузить новую страницу. Время замещения страниц сокращается.

б) Отложенное вытеснение: страница помещается в буфер кандидатов на вытеснение (своего рода второй шанс). Если к ней происходит обращение, пока она в буфере (т. наз. «мягкое» страничное прерывание), то нет потерь на загрузку с диска. Так преодолевается основной недостаток FIFO

NB: оба буфера а) и б) можно рассматривать как кэш страничного обмена

Приемы улучшения алгоритмов (2)

2. Буферизация с помощью "упреждающих" действий

в) Упреждающая запись модифицированной страницы в область свопинга (на всякий случай; обычно в низкоприоритетном потоке) и сброс M:=0

г) Кластеризация страниц: вместе с новой страницей загружаются несколько соседних – *кластер страниц*

д) Опережающая подкачка страниц (prepaging) перед запуском процесса или на ходу - на основе предсказания (для страниц данных)

Заключение

1. Память имеет несколько уровней иерархии, на каждом из которых свой экономически оправданный компромисс между емкостью и скоростью
2. В управлении памятью – две основные задачи:
   - отображение виртуальных адресов на физические
   - управление обменом с диском при виртуальной памяти
3. Рост емкости памяти идет параллельно с увеличением размеров программ и объемов данных, так что памяти постоянно не хватает
4. Способы борьбы с нехваткой памяти – свопинг, оверлейная структура и виртуальная память
5. Страничная/сегментная организация памяти выполняет две функции:
   - борьба с фрагментацией основной памяти
   - основа виртуальной памяти
6. Страничная организация более распространена; сегментная используется только вдобавок к страничной