Операционные системы

Курс лекций для гр. 4057/2

Лекция №8

Вопросы к лекции 7

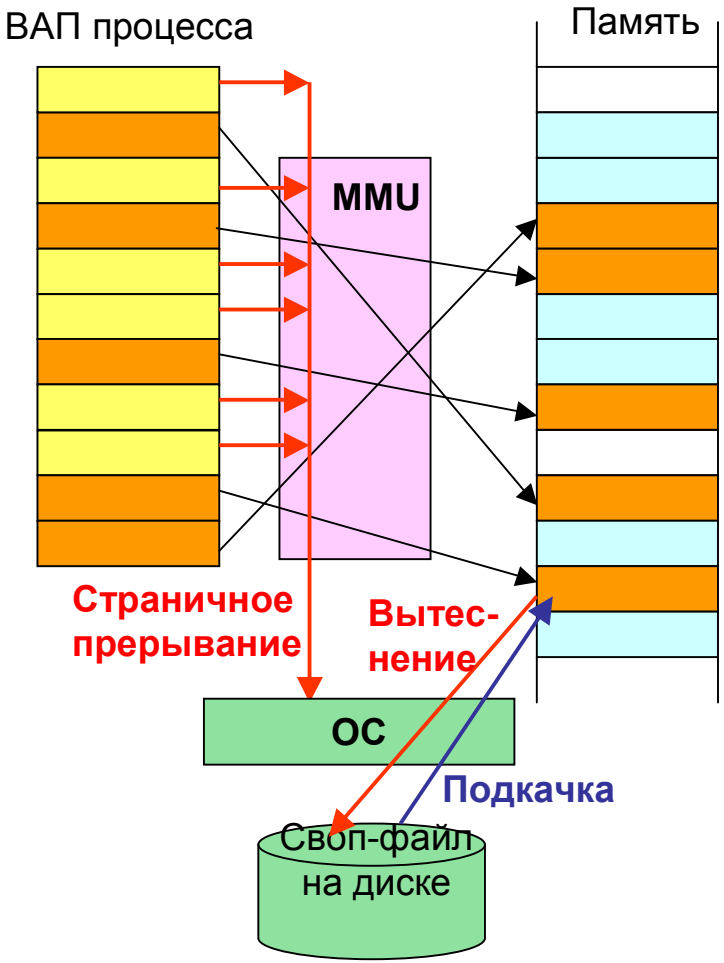
1. Чем определяется размер ФАП ?
2. Зачем транзитные программы ОС обычно делают абсолютными, а не перемещаемыми ?
3. В чем преимущества динамического связывания и подгрузки подпрограмм по сравнению со статическим связыванием ?
4. В чем недостатки MFT ?
5. Почему внутренняя фрагментация при страничном распределении обычно больше, чем при непрерывном? Каков в среднем ее размер ?
6. Почему размер страницы в байтах всегда = 2^N , где N – целое? Что будет, если сделать не так? Например, если вместо 4096-байтной страницы сделать ее размером 5000 или 4000 байт?
7. Каково максимальное число страниц, которое может содержать процесс в 32-разрядной архитектуре со страницами размером 4 Кб ? Сколько элементов в таблице страниц для процесса размером 5 Мб ?
8. Пусть время доступа к памяти = 100 нс, а к кэш-памяти = 20 нс. Каково эффективное среднее время доступа к памяти со страничной организацией и TLB при вероятностях попадания 80% и 98% ?
9. Обоснуйте каждое из преимуществ - ответьте на вопрос: почему?
10. Какие проблемы чисто сегментной и чисто страничной организации виртуальной памяти снимаются в смешанной схеме?

Содержание

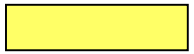
Виртуальная память

Цель: увеличить суммарный размер параллельно выполняемых процессов при ограниченном ФАП

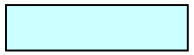
Идея: организовать основную память как кэш для диска



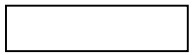
активная страница, находится в памяти



неактивная страница, хранится на диске



активная страница другого процесса



пустой кадр в памяти

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 80x86)

Бит активности **A** - флаг присутствия страницы в памяти: **A=1** **A=0**

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.

У неактивной страницы структура другая – она содержит адрес страницы на диске

Обработка страничного прерывания:

- Если есть свободная память, загрузить в нее запрошенную страницу
- Если свободной памяти нет, то делается *замещение* страниц) :
  1) найти страницу, подлежащую вытеснению из памяти
  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

Алгоритмы замещения страниц

Цель: минимизировать частоту страничных прерываний.

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

Такая идеальная стратегия называется OPT (оптимальная) (Вопрос 1)

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

   - Очень правдоподобно предположение, что LRU наиболее близка OPT
   - Недостаток: работает плохо, если процесс циклически обращается к последовательности страниц, которые не помещаются в память одновременно

(Вопрос 2)

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) (Вопрос 3)
- 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

  (Вопрос 4)

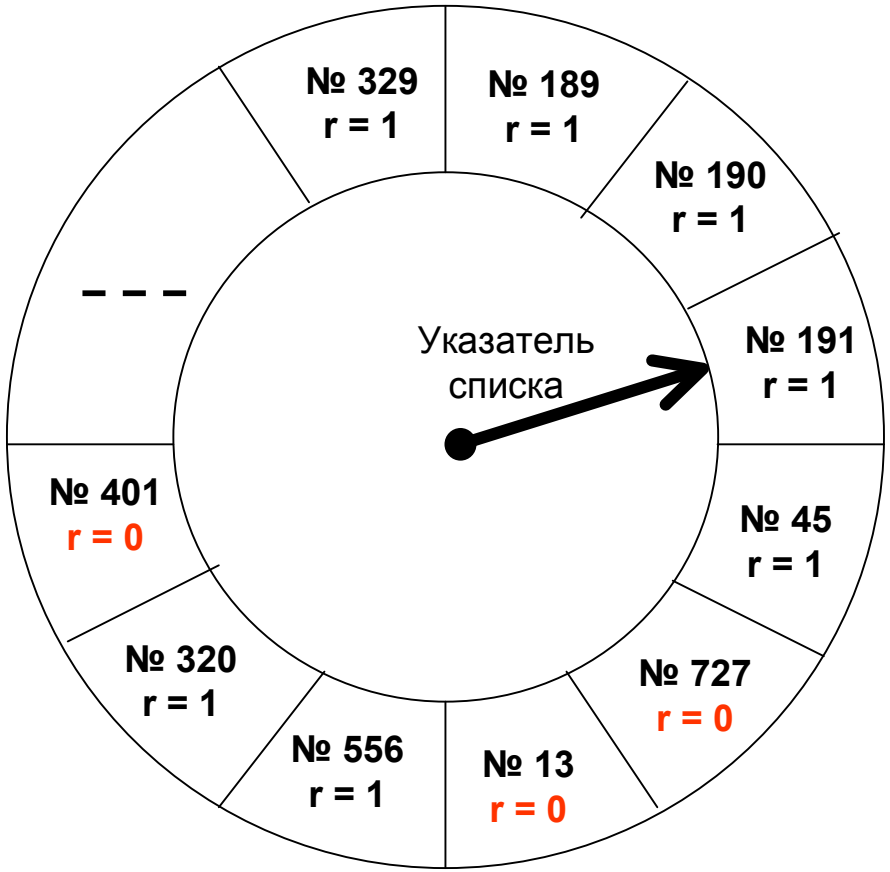
Аппроксимации LRU (4)

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

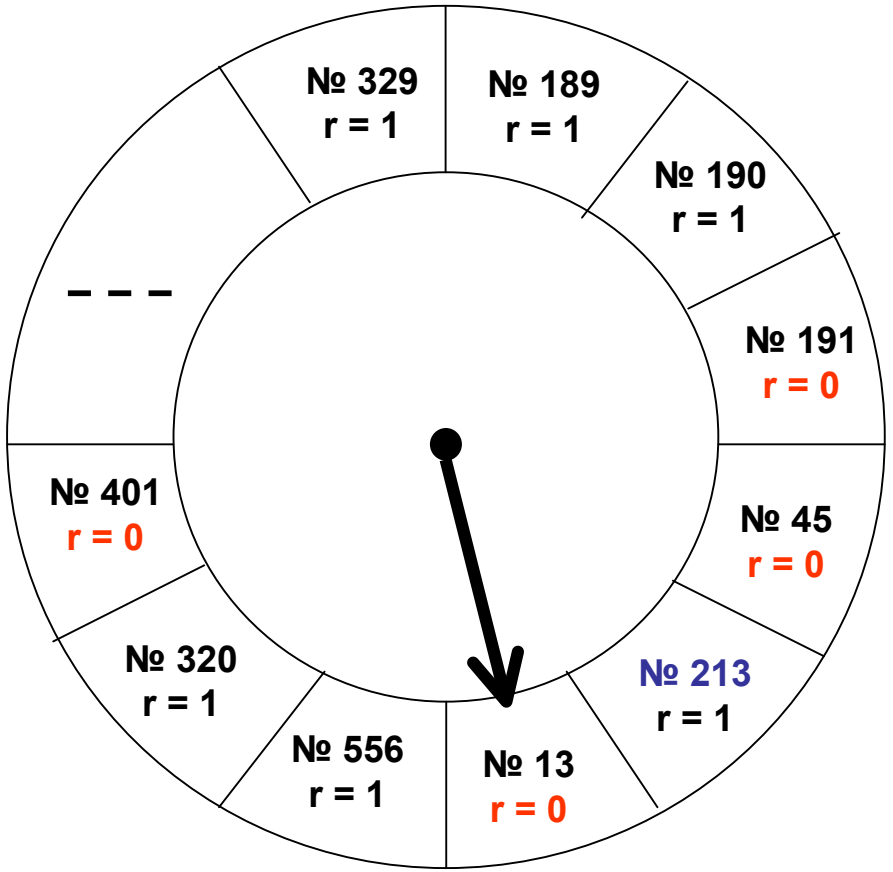
- Кольцевой список номеров активных страниц; с каждым номером связан *ссылочный (reference)* бит **r** – счетчик использования
- При каждом обращении к странице ее **r** устанавливается в 1
- При каждом страничном прерывании ОС проверяет **r** очередной страницы:
  - если **r = 0**, то эта страница замещается 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



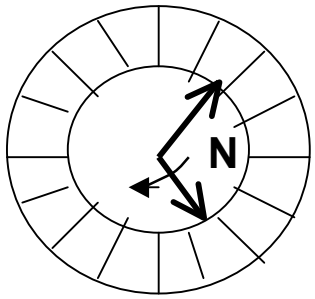
Страница № 213 заместила страницу № 727

Реализация часового алгоритма
в одной из версий Linux

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

Варианты часового алгоритма в версиях Unix/Linux

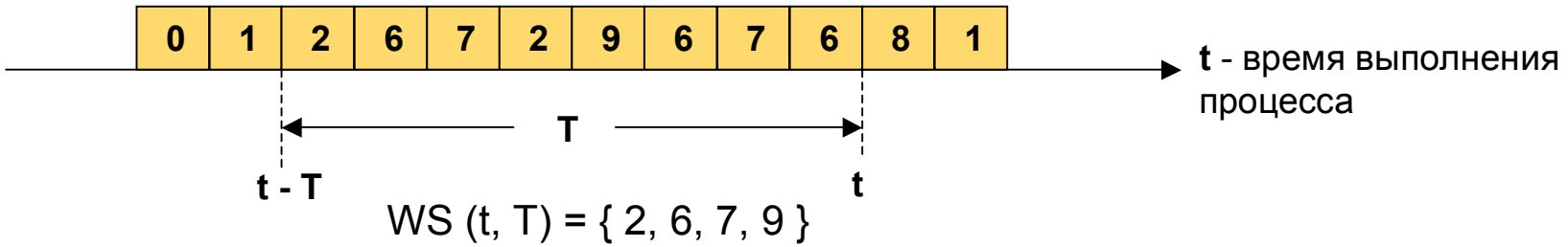
- Двухбитовый счетчик (как в NUR): R и M
  - при M=1 страница не вытесняется
- 8-битовый счетчик:
  - ++ при каждом обращении к странице
  - регулярное уменьшение счетчика у всех страниц (как в алг. “Старение”)
- Две стрелки, вращающиеся синхронно:
  - первая обнуляет счетчик r
  - вторая с отставанием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



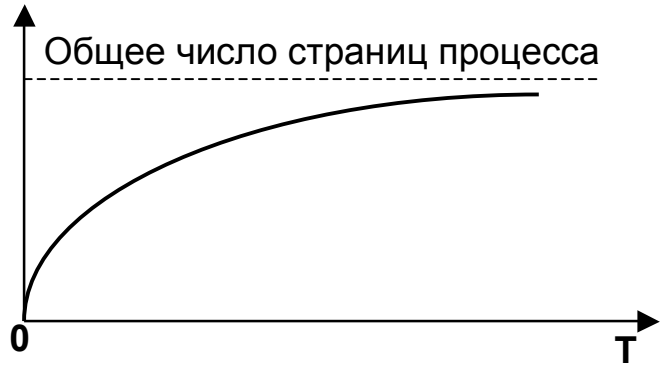
Алгоритм рабочего набора

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)



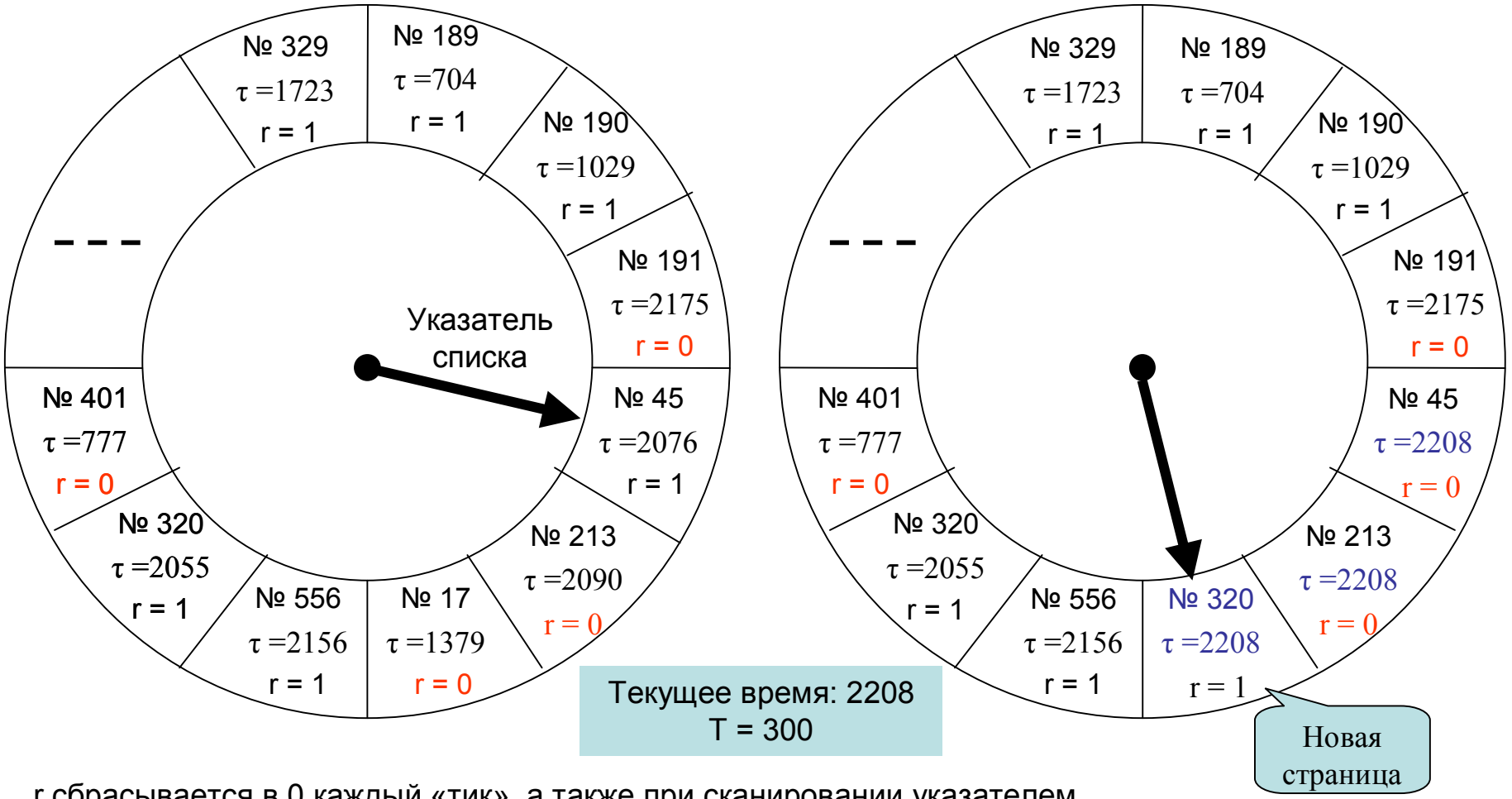
|WS| - число страниц рабочего набора



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) 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

б) 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.

NB: оба буфера а) и б) можно рассматривать как кэш страничного обмена с FIFO-стратегией

Приемы улучшения алгоритмов (2)

2. Буферизация с помощью "упреждающих" действий

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

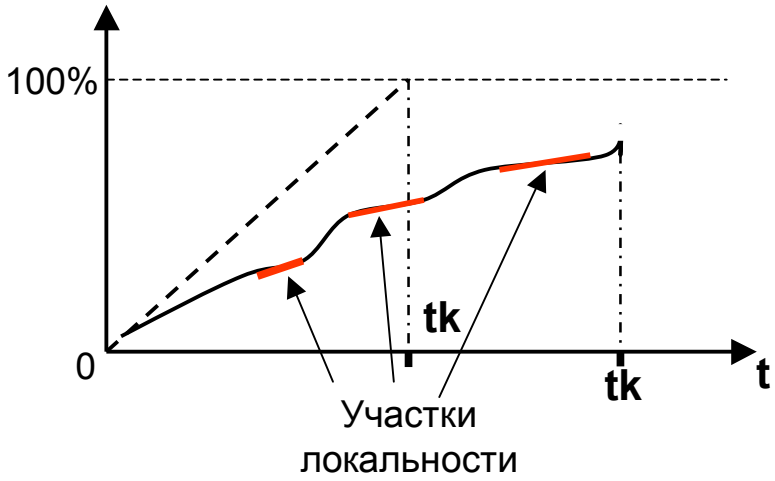
б) Опережающая подкачка страниц (prepaging) перед запуском процесса или на ходу - на основе предсказания их запроса

в) Кластеризация страниц: вместе с новой страницей на всякий случай загружаются несколько соседних – *кластер страниц*

Динамика страничного обмена

Обращение к страницам процесса по мере его выполнения описывается монотонно неубывающей функцией:

S - доля страниц процесса, к которым было обращение



– – – – – – – линейная программа (на самом деле ступенчатый график)

———— программа с разветвлениями и циклами

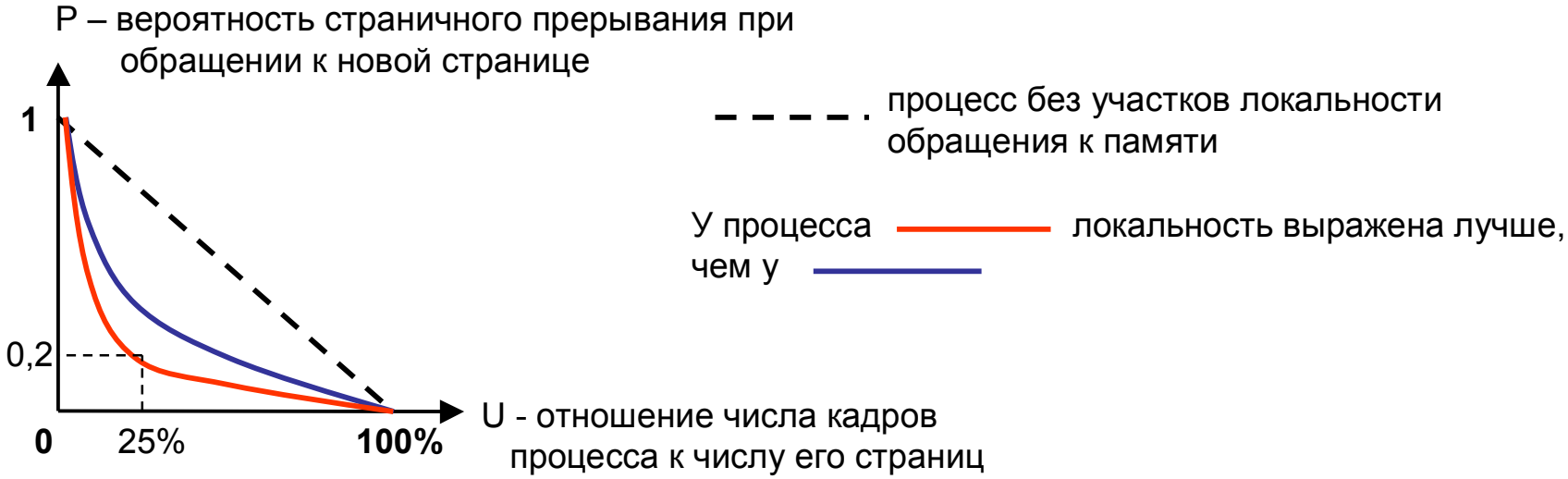
tk – время завершения процесса

Явление *локальности обращения к памяти*:
очередное обращение – чаще всего к странице из текущего рабочего набора; это приводит к горизонтальному или очень пологому участку на графике

Временная причина локальности - циклы, пространственная – массивы данных, неоднократно просматриваемые

Локальность - положительное явление; существуют компиляторы и сборщики, улучшающие локальность кода

Явление пробуксовки



Пробуксовка (thrashing) возникает, когда б**о**льшая часть времени выполнения процесса тратится на страничный обмен, и общая производительность системы падает

Считается, что для большинства процессов при U > 25% пробуксовка мала и P < 0,2

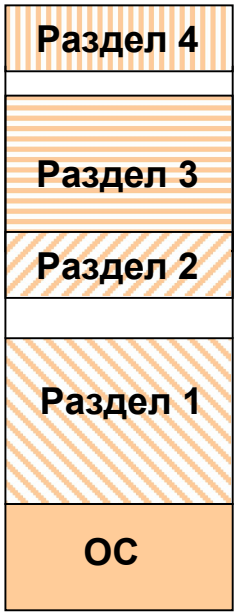
Выделение кадров и замещение страниц при мультипрограммировании (1)

A. Управление размером *резидентного множества* страниц процесса (т.е., размером выделяемой ему памяти - числа кадров - аналога размера раздела памяти при непрерывном распределении):

A1. Фиксированное распределение: фиксированное число кадров памяти выделяется процессу в момент его начальной загрузки

A2. Переменное распределение: число кадров может динамически изменяться в ходе выполнения процесса, в соответствии с его запросами

Преимущества A2 очевидны – подобны преимуществам MVT перед MFT



Выделение кадров и замещение страниц при мультипрограммировании (2)

Б. Область видимости, или стратегия замещения:

Б1. Локальная: вытесняется страница только данного процесса

Б2. Глобальная: кандидаты на вытеснение принадлежат всем процессам

Три комбинации А и Б (А1&Б2 несовместимы):

А1&Б1 – фиксированное распределение & локальная область видимости

Недостатки те же, что у MFT; применяется редко

А2&Б2 – переменное распределение & глобальная область видимости (Linux)

Плюсы: гибкость, возможность подстройки к процессам различного размера

Минус: опасность пробуксовки из-за:

- монополизации памяти суперактивным процессом
- конфликтов интересов процессов при их циклической диспетчеризации (RR)

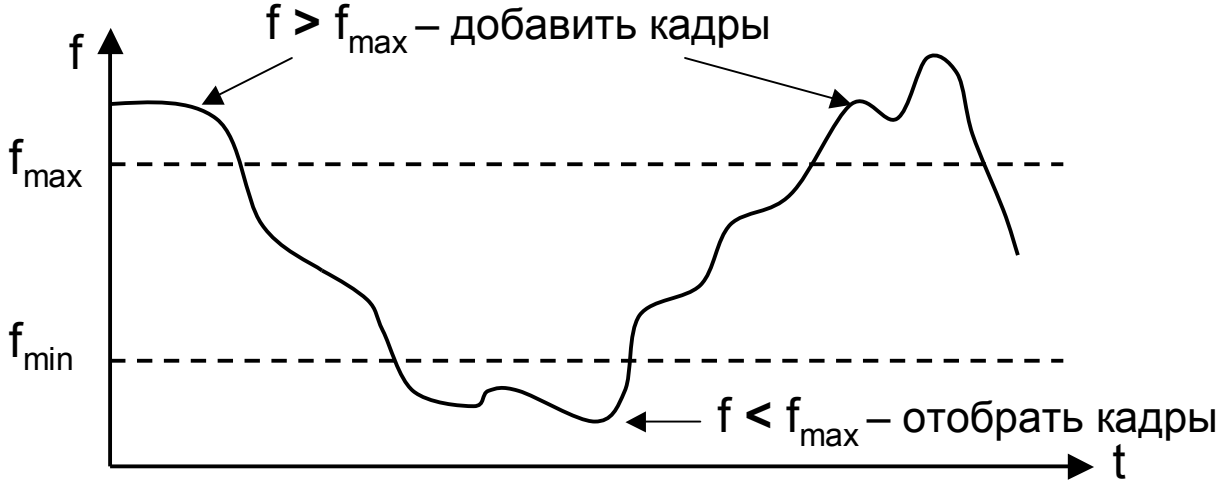
А2&Б1 - переменное распределение & локальная область видимости (Win2k) - недостатки предыдущего варианта устраняются, если время от времени корректировать размер резидентного множества

NB: Для части страниц действует запрет на вытеснение: ядро ОС, буферы вв-вы и т.п.

Адаптивная подстройка размера резидентного множества страниц процесса

Как динамически определять, насколько количество кадров процесса близко к оптимальному ?

Приблизительная оценка - по частоте страничных прерываний f:
высокая – памяти нехватает, слишком низкая – памяти избыток



(Вопрос 5)

Пробуксовка мала, если

$f_{max} \geq 1/T_{странич.\ обмена}$

Если при этом

$1/T_{странич.\ обмена} \geq f_{min}$,

то $1/f \sim T_{странич.\ обмена}$

Одно из 1 млн обращений к памяти приводит к страничному прерыванию

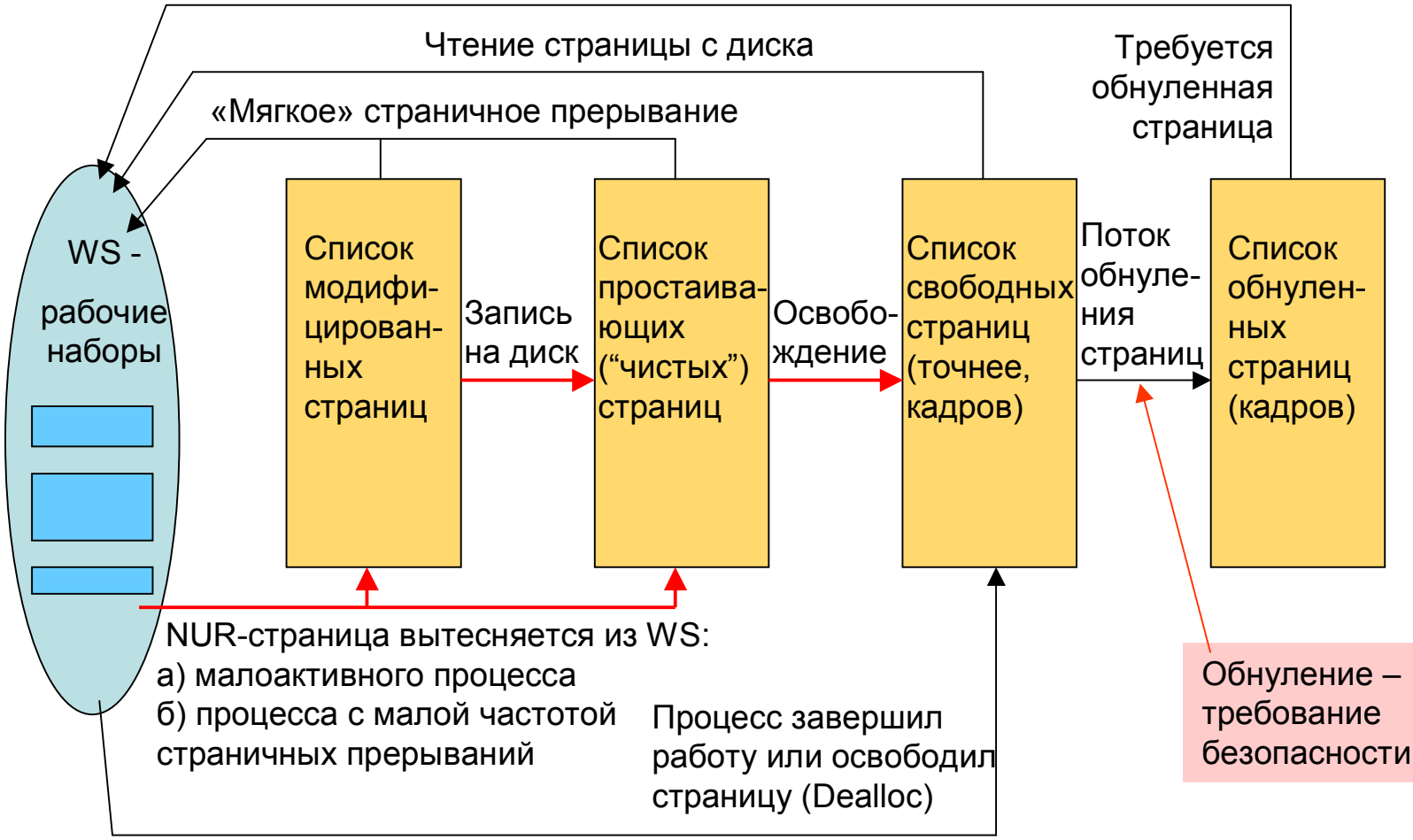
Если сделать f < fmax для всех процессов одновременно не удается, то число активных процессов следует сократить, приостановив некоторые из них (т.е., снизить текущий коэффициент мультипрограммирования)

Виртуальная память Windows

- Используется двухуровневая страничная организация Intel 80x86 (сегментный уровень используется не для виртуальной памяти, а для защиты)
- Каждое приложение имеет 4 Гб ВАП, из которых верхние 2 Гб зарезервированы для ОС и используются совместно всеми процессами (кроме таблиц страниц, индивидуальных для каждого процесса)
  - благодаря этому при обращении к системному вызову не нужно переключать таблицу страниц, а нужно только переключиться в режим ядра
- Управление резидентным множеством страниц процесса – переменное распределение с локальной областью видимости
- При нехватке свободной памяти диспетчер ВП вытесняет страницы из резидентных множеств страниц активных процессов, уменьшая их размеры. Алгоритм вытеснения:
  - для однопроцессорной системы - NUR
  - для многопроцессорной системы – FIFO (Вопрос 6)
- Min и Max количества кадров вначале одинаковы для каждого процесса, но могут меняться со временем
- Когда процесс запускается, в памяти нет ни одной его страницы (т.е., опережающая подкачка отсутствует)

(Вопросы 7, 8)

Win 2K: база данных PFN
(Page Frame Numbers)



Рабочий набор – то же, что и резидентное множество

Все списки – это FIFO-очереди, они пополняются по переходам периодически, по результатам сканирования потоком *свопера*

Виртуальная память Unix/Linux

- Сегментно-страничная организация
- Одна таблица сегментов на процесс
- На 64-битовом процессоре - трехуровневая страничная схема
- На 32-битовом процессоре:
  - один (средний) уровень не используется
  - каждый процесс получает 3 Гб ВАП для себя и 1 Гб для ядра ОС
- Алгоритм замещения страниц – глобальный часовой с отложенным вытеснением и улучшениями:
  - буфер свободных страниц - 25% общего объема памяти
  - периодическое сканирование списка страниц страничным демоном с целью поиска и вытеснения долго неиспользуемых страниц
  - опережающая подкачка страниц при старте процесса (но не в Linux)

Заключение

1. Виртуальная память – способ превращения основной памяти в кэш для диска
2. Виртуальная память требует существенной аппаратной поддержки (напр., счетчиков), которая во многом определяет возможные алгоритмы вытеснения
3. Наиболее распространенные алгоритмы вытеснения страниц – NUR-аппроксимация LRU, часовой и FIFO (все с улучшениями - с буферизацией замещения страниц)
4. Разнообразие алгоритмов вытеснения страниц для разных версий Unix/Linux велико и продолжает увеличиваться
5. Динамическое изменение размеров резидентного множества страниц процесса – типичное решение, аналогичное MVT
6. Настройка параметров виртуальной памяти для достижения лучшей производительности – задача системных программистов

Вопросы к лекции 8

1. В каких ситуациях и для каких страниц стратегия OPT достижима?
2. Решите задачи и докажите утверждение в Приложении (слайд 28).
3. Что будет, если период между сбросами битов в ноль очень мал ? Очень велик ?
4. Почему страница с R=1, M=0 приоритетнее для вытеснения, чем с R=0, M=1 ?
5. Пусть время доступа к активной странице = 200 нс, время подкачки отсутствующей страницы = 20 мс (TLB отсутствует). Насколько мала должна быть относительная частота страничных прерываний f, чтобы эффективное (т.е, среднее) время доступа к памяти не превышало 220 нс, т.е. было только на 10% больше ? (f оценивайте отношением числа обращений, приводящих к страничным прерываниям, к общему числу обращений к памяти.)
6. Почему для многопроцессорной системы NUR - не рационально?
7. Пусть несколько приложений выполняются под Windows некоторое время, а потом все они одновременно закрываются и перезапускаются снова. Почему в первое время после перезапуска общая производительность системы обычно ниже, чем до него?
8. Если запустить интерактивное мультимедиа-приложение (напр., игру), не закрыв другие приложения, то в первое время производительность его будет низка (игра будет «тормозить», особенно, если объем графических данных - большой). Объясните причину этого эффекта и предложите программный способ преодоления этого недостатка под Windows.

Приложение

Подсчитайте число страничных прерываний для различных алгоритмов вытеснения страниц, числа доступных кадров и последовательностей обращения к страницам, считая, что все кадры первоначально пусты.

№ кадра	Последовательность обращения к страницам →				
	A	B	C	B	D
1	A	A	A	A	D
2		B	B	B	B
3			C	C	C
Странич. прерыв.	+	+	+		+

Для этого заполните таблицы соответствия виртуальных страниц кадрам памяти по типу приведенной, составленной для алгоритма FIFO, трех доступных кадров памяти и последовательности обращения к четырем страницам: A, B, C, B, D

Требуется короткий ответ: общее число страничных прерываний для каждого варианта задачи

Задача №1. Алгоритмы OPT, FIFO, LRU; 3 кадра; последов. A, B, C, A, B, D, A, D, B, C, B

Задача №2. Алгоритм LRU; 3 кадра; последов. A, B, C, D, A, B, C, A, B, C, D

Задача №3. Алгоритм FIFO и LRU; 3 и 4 кадра; последов. A, B, C, D, A, B, E, A, B, C, D, E

Задание: Докажите, что при LRU увеличение числа кадров *всегда* уменьшает число страничных прерываний