7.1 Алгоритмы замещения страниц памяти

Идеальный алгоритм заключается в том, что бы выгружать ту страницу, которая будет запрошена позже всех.

Но этот алгоритм не осуществим, т.к. нельзя знать какую страницу, когда запросят. Можно лишь набрать статистику использования.

7.1.1 Алгоритм NRU (Not Recently Used - не использовавшаяся в последнее время страница)

Используются биты обращения (R-Referenced) и изменения (M-Modified) в таблице страниц.

При обращении бит R выставляется в 1, через некоторое время ОС не переведет его в 0.

M переводится в 0, только после записи на диск.

Благодаря этим битам можно получить 4-ре класса страниц:

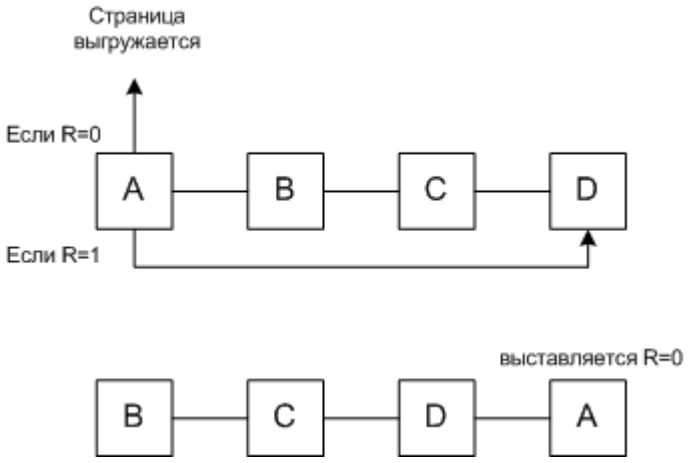
1. не было обращений и изменений (R=0, M=0)
2. не было обращений, было изменение (R=0, M=1)
3. было обращение, не было изменений (R=1, M=0)
4. было обращений и изменений (R=1, M=1)

7.1.2 Алгоритм FIFO (первая прибыла - первая выгружена)

Недостаток заключается в том, что наиболее часто запрашиваемая страница может быть выгружена.

7.1.3 Алгоритм "вторая попытка"

Подобен FIFO, но если R=1, то страница переводится в конец очереди, если R=0, то страница выгружается.



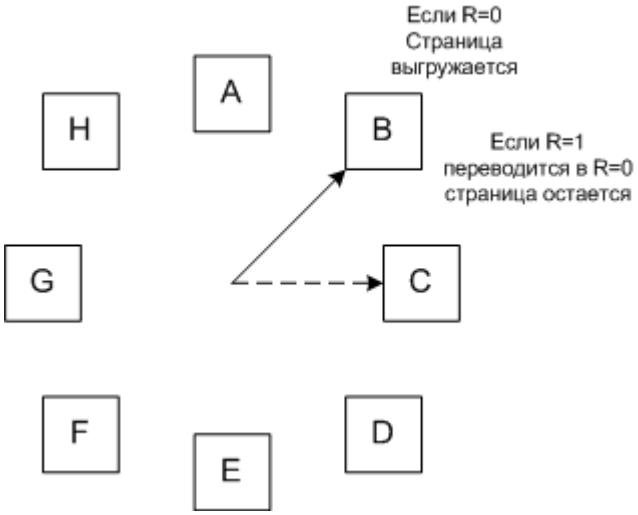
Алгоритм "вторая попытка"

В таком алгоритме часто используемая страница никогда не покинет память.

Но в этом алгоритме приходится часто перемещать страницы по списку.

7.1.4 Алгоритм "часы"

Чтобы избежать перемещения страниц по списку, можно использовать указатель, который перемещается по списку.



Алгоритм "часы"

7.1.5 Алгоритм LRU (Least Recently Used - использовавшаяся реже всего)

Первый метод:

Чтобы реализовать этот алгоритм, можно поддерживать список, в котором выстраивать страницы по количеству использования. Эта реализация очень дорога.

Второй метод:

В таблице страниц добавляется запись - счетчик обращений к странице. Чем меньше значение счетчика, тем реже она использовалась.

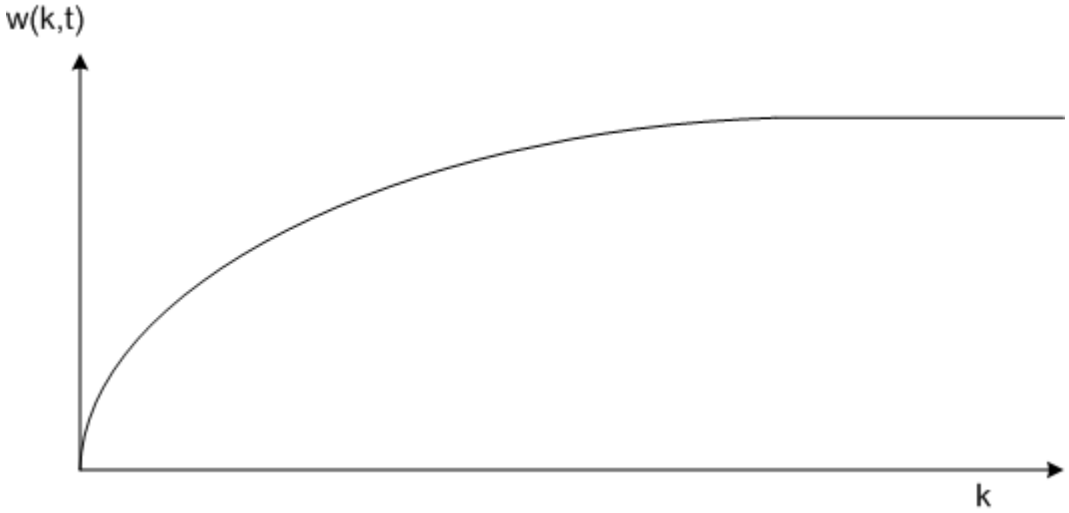
7.1.6 Алгоритм "рабочий набор"

Замещение страниц по запросу - когда страницы загружаются по требованию, а не заранее, т.е. процесс прерывается и ждет загрузки страницы.

Буксование - когда каждую следующую страницу приходится процессу загружать в память.

Чтобы не происходило частых прерываний, желательно чтобы часто запрашиваемые страницы загружались заранее, а остальные подгружались по необходимости.

Рабочий набор - множество страниц (к), которое процесс использовал до момента времени (t). Т.е. можно записать функцию w(k,t).



Зависимость рабочего набора w(k,t) от количества запрошенных страниц

Т.е. рабочий набор выходит в насыщение, значение w(k,t) в режиме насыщения может служить для рабочего набора, который необходимо загружать до запуска процесса.

Алгоритм заключается в том, чтобы определить рабочий набор, найти и выгрузить страницу, которая не входит в рабочий набор.

Этот алгоритм можно реализовать, записывая, при каждом обращении к памяти, номер страницы в специальный сдвигающийся регистр, затем удалялись бы дублирующие страницы. Но это дорого.

В принципе можно использовать множество страниц, к которым обращался процесс за последние **t** секунд.

Текущее виртуальное время (Tv) - время работы процессора, которое реально использовал процесс.

Время последнего использования (Told) - текущее время при R=1, т.е. все страницы проверяются на R=1, и если да то текущее время записывается в это поле.

Теперь можно вычислить возраст страницы (не обновления) **Tv-Told,** и сравнить с **t**, если больше, то страница не входит в рабочий набор, и страницу можно выгружать.

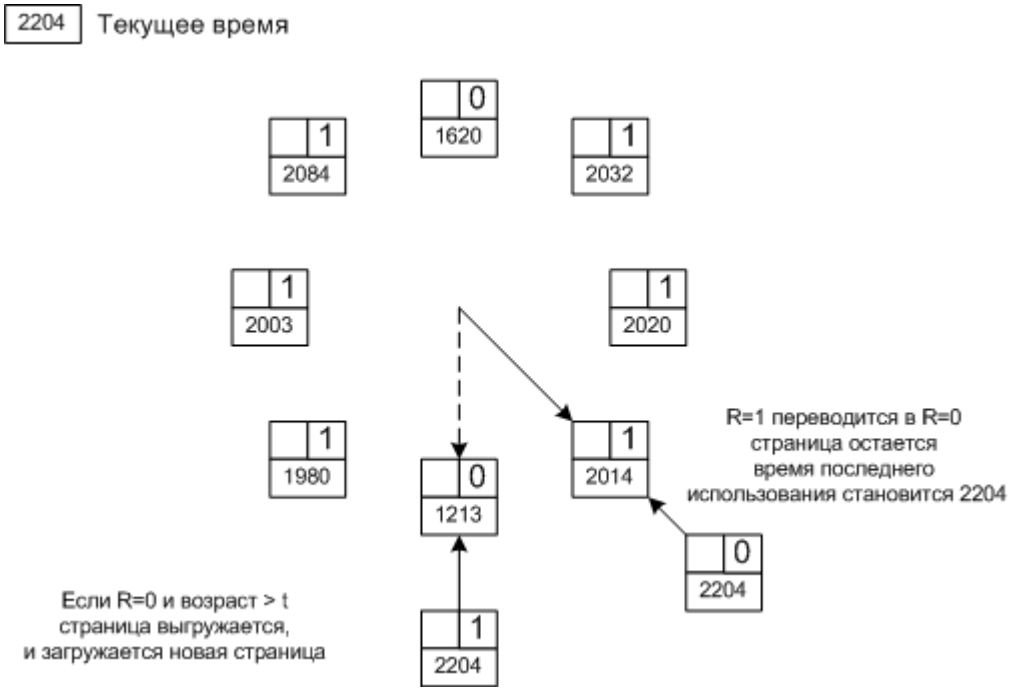
Получается три варианта:

- если R=1, то текущее время запоминается в поле время последнего использования
- если R=0 и возраст > t, то страница удаляется
- если R=0 и возраст =< t, то эта страница входит в рабочий набор

7.1.7 Алгоритм WSClock

Алгоритм основан на алгоритме "часы", но использует рабочий набор.

Используются битов R и M, а также время последнего использования.



Работа алгоритма WSClock

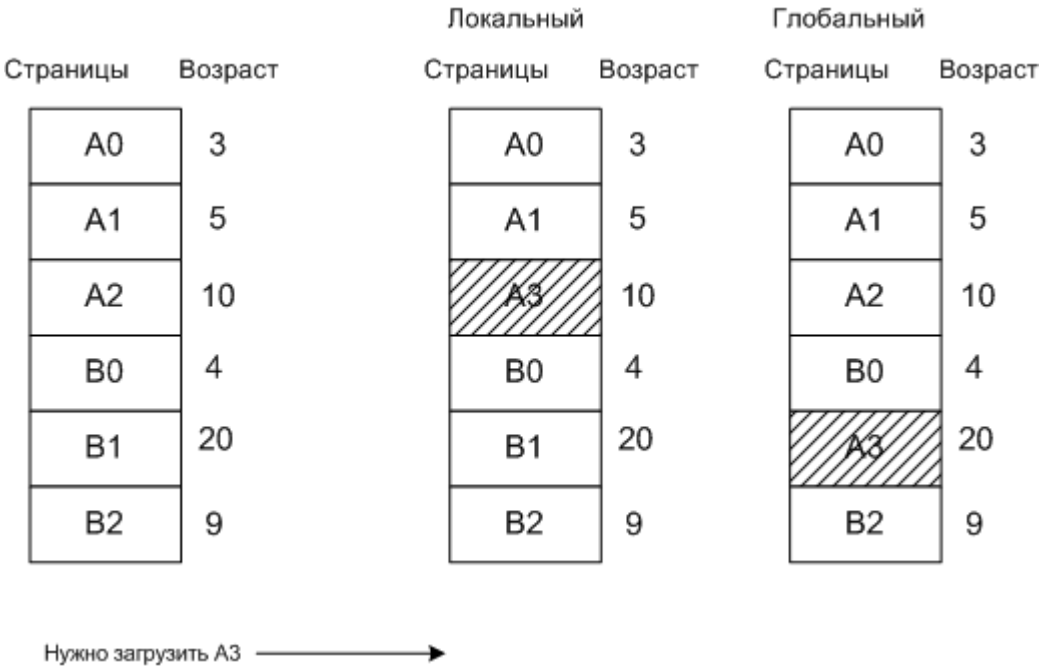
Это достаточно реальный алгоритм, который используется на практике.

7.2 Распределение памяти

7.2.1 Политика распределения памяти

Алгоритмы замещения бывают:

- локальные
- глобальные



Пример глобального и локального алгоритма

В целом глобальный алгоритм работает лучше.

Можно поровну распределять страничные блоки между процессами.

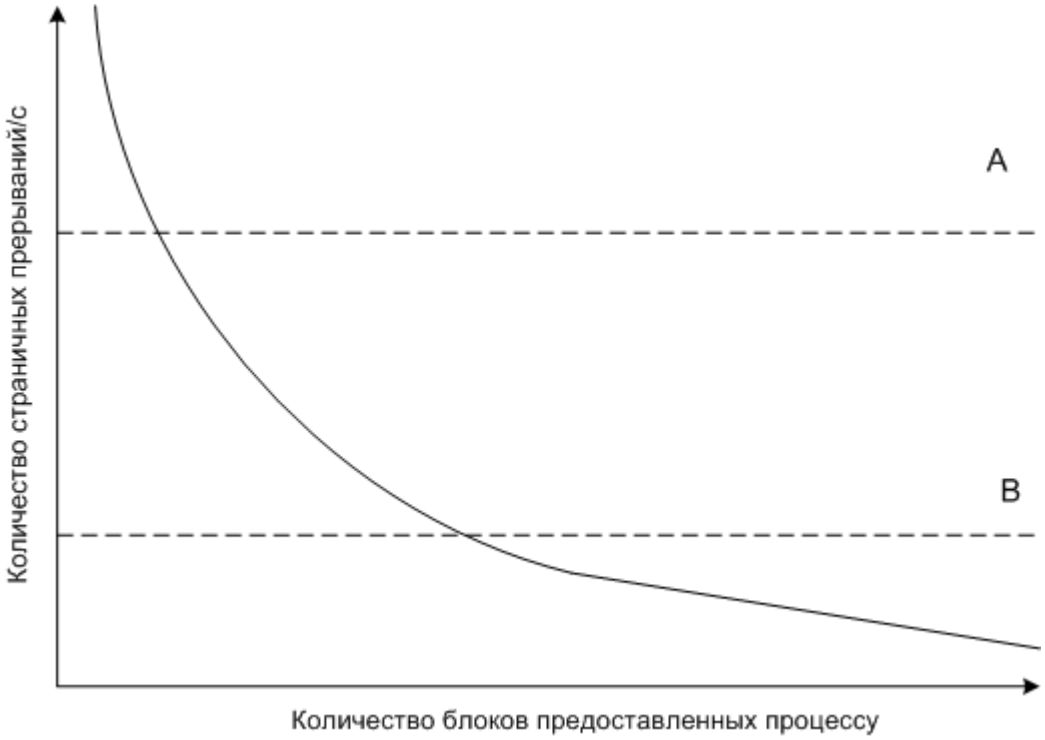
Такой подход справедлив, но не эффективен, т.к. процессы разные.

Можно распределять страничные блоки между процессами, в зависимости от размеров процесса

Размер процесса динамически меняется, поэтому определить размер динамически сложно.

Частота страничных прерываний - может служить показателем потребности процесса в страницах.

Чем больше частота, тем больше памяти необходимо процессу.



Зависимость частоты страничных прерываний от размеров памяти предоставленной процессу

Если частота стала ниже линии B, то памяти процессу предоставлено слишком много.

Если частота стала выше линии A, то памяти процессу предоставлено слишком мало.

Если всем процессам не хватает памяти (происходит пробуксовка), то производится выгрузка какого то процесса на диск.

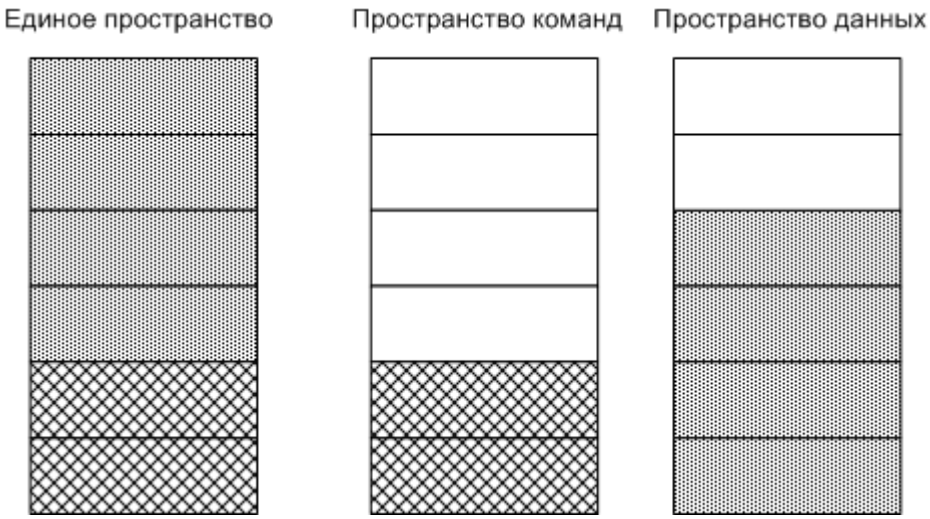
7.2.2 Размеры страниц

Есть два крайних случая:

- Маленькие страницы - улучшает распределение памяти, но увеличивает таблицу и частые переключения уменьшают производительность.
- Большие страницы - наоборот.

7.2.3 Совместно используемые страницы

Отдельные пространства команд и данных



Пример разделения пространства команд и данных

Совместно используемые страницы

Два процесса могут содержать в таблицах страниц указатели на общие страницы. В случае разделения пространств команд и данных это легко реализуется. Эти данные используются в режиме чтения.

В UNIX, когда создается дочерний процесс, у родительского и дочернего процесса общее пространство данных, и только если один из процессов попытается изменить данные, происходит прерывание и создание копии этой страницы, если записи не происходит, то оба процесса продолжают работать с общей памятью. Это приводит к экономии памяти.

7.2.4 Политика очистки страниц

Лучше всегда держать в запасе свободные блоки, освобождая их заранее, чем при нехватке памяти, искать и освобождать их.

Страничный демон - программа, периодически проверяющая состояние памяти, если занято много блоков, то производит выборочную выгрузку страниц.

7.3 Особенности реализации в UNIX

В UNIX системах последовательность запуска процессов, следующая:

процесс 0 - это свопер

процесс 1 - это init

процесс 2 - это страничный демон

Страничный демон просыпается каждые 250мс, и проверяет количество свободных страничных блоков, если их меньше 1/4 памяти, то он начинает выгружать страницы на диск. Он использует модифицированный алгоритм часов, и он является глобальным (т.е. он не различает, какому процессу принадлежит страница).

Каждые несколько секунд свопер проверяет, есть ли на диске готовые процессы для загрузки в память для выполнения. При этом сам код программы в своп-файле не сохраняется, а подкачивается непосредственно из файла программы.

В LUNIX системе нет предварительной загрузки страниц и концепции рабочего набора.

Тексты программ и отображаемые файлы подгружаются прямо из файлов расположенных на диске.

Все остальное выгружается в раздел свопинга или файлы свопинга (их может быть от 0 до 8).

Алгоритм выгрузки страниц основан на страничном демоне (kswapd), он активизируется раз в секунда и проверяет достаточно ли свободных страниц. Демон может быть активизирован и принудительно, при не хватке памяти.

Демон состоит из трех процедур:

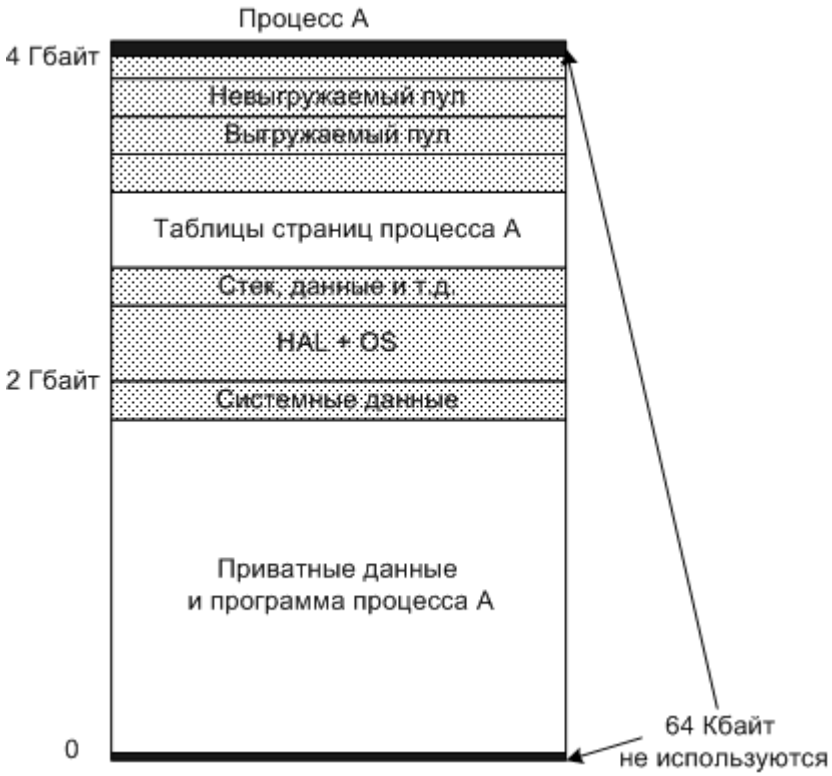
- В первой используется алгоритм часов, она ищет редко используемые страницы страничного кэша и буферного кэша файловой системы.
- Вторая процедура ищет совместно редко используемые страницы.
- Третья ищет редко используемые страницы одиночных пользователей. Сначала сканируются страницы у того процесса, у которого их больше всего.

В LINUX есть еще один демон - bdflush. Он регулярно просыпается и проверяет, не превысило ли определенное значение количество измененных страниц, если да то он начинает их принудительно сохранять на диск.

7.4 Особенности реализации в Windows

В Windows системах сегментация (следующая лекция) не поддерживается. Поэтому каждому процессу выделяется виртуальное адресное пространство в 4 Гбайт (ограничение 32 разрядов). Нижние 2 Гбайт доступны для процесса, а верхние 2 Гбайт отображаются на память ядра. В Advanced server и Datacenter server процесс может использовать до 3 Гбайт.

Страницы имеют фиксированный размер (на процессорах Pentium 4 Кбайт, на Itanium 8 или 16 Кбайт) и подгружаются по требованию.



Конфигурация виртуального адресного пространства Windows

Белым цветом выделены области приватных данных процесса.

Затемнены области, совместно используемые всеми процессами.

Области в 64 Кбайт в начале и в конце, используются для защиты виртуального адресного пространства процесса, при попытке чтения или записи в эти области будет вызвано прерывание.

Системные данные содержат указатели и таймеры, доступные на чтение другим процессам.

Отображение верхней части на память ядра, позволяет при переключении потока в режим ядра не менять карту памяти.

У страниц есть три состояния:

- свободное - не используется
- фиксированное - данные отображены в странице
- зарезервированное - зарезервировано, но не занято данными (при создании потока)

Файлы свопинга может быть до 16, разделов свопинга нет. В файлах свопинга хранятся только изменяемые страницы.

Опережающая подкачка в Windows не используется.

В Windows используется понятие рабочий набор.

Страничный демон в Windows состоит из :

- менеджера балансового множества - проверяет, достаточно ли свободных страниц.
- менеджера рабочих наборов - который исследует рабочие наборы и освобождает страницы.

Также в Windows есть следующие демоны:

- свопер-демон
- демон записи отображенных страниц - запись в отображенные файлы
- демон записи модифицированных страниц