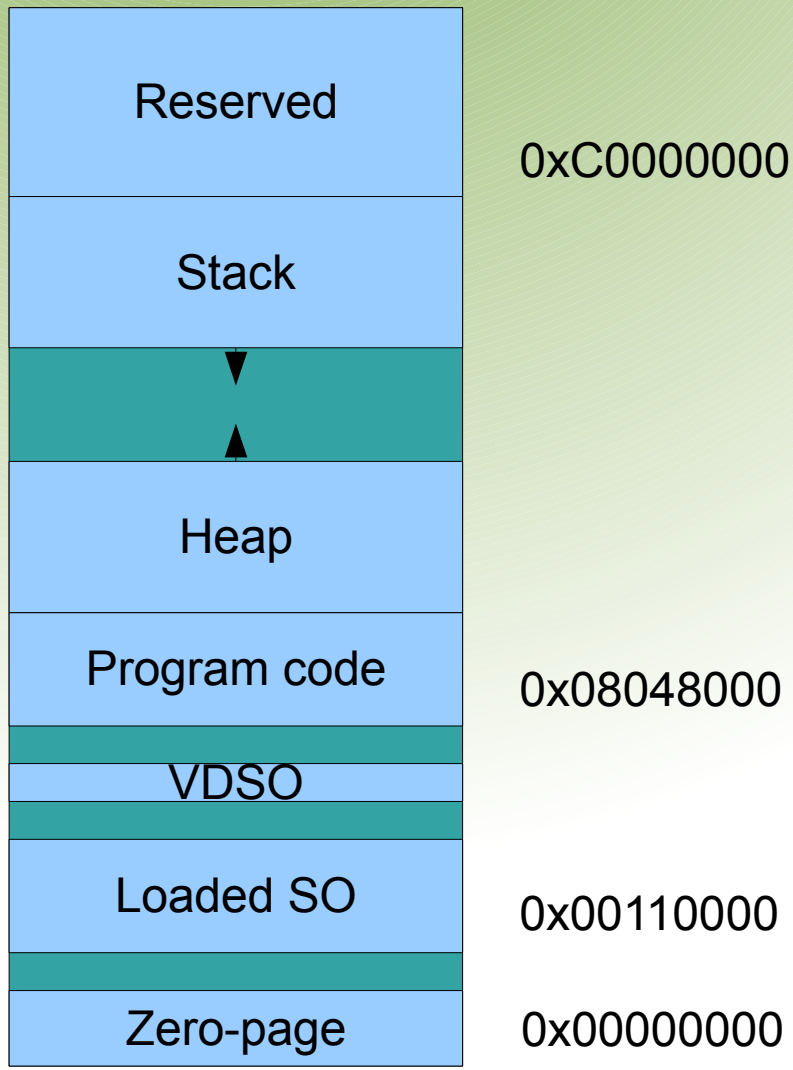
Управление памятью

- Управление памятью с точки зрения процесса
  - Адресное пространство процесса
  - Управление адресным пространством
  - Отображаемые в память файлы
  - Динамические библиотеки

Адресное пространство процесса

- Пример: 32-битное виртуальное адресное пространство, макс. 4GiB
- Ядро ОС резервирует часть адресного пространства для своих нужд, обычно 3GiB/1GiB или 2GiB/2GiB
- Если размер физической памяти не превышает 1GiB, то верхняя скрытая часть адресного пространства процесса может быть напрямую отображена на физическую память

Адресное пространство процесса

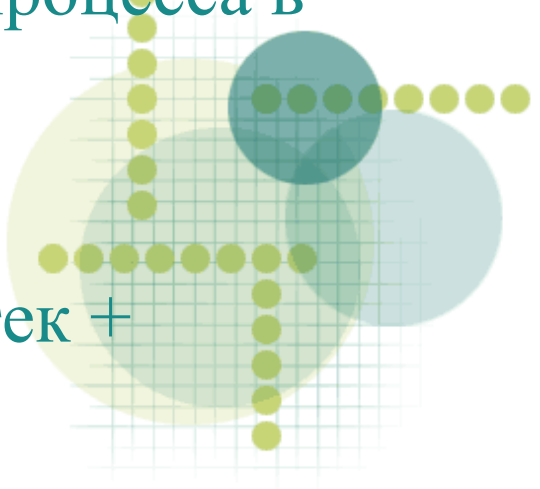


- Нулевая страница — защита от обращений по указателю NULL
- Стек расширяется вниз автоматически
- Куча растет вверх по запросу
- Текущее состояние карты памяти: /proc/${PID}/maps

Параметры памяти процесса

```
[~]$ps v
  PID TTY      STAT   VSZ   MAJFL   TRS   DRS   RSS %MEM COMMAND
 8241 pts/0    Ss     6912      0   716  6195  2756  0.0 -bash
```

- VSZ — размер виртуального адресного пространства
- %MEM — процент занятой физической памяти
- RSS (resident set size) — размер страниц процесса в ОЗУ
- TRS — размер области кода процесса
- DRS — размер области данных (куча + стек + библиотеки) процесса



Ограничения адресного пространства

- Команда ulimit — установка ограничений процесса

```
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 57326
max locked memory       (kbytes, -l) 32
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 1024
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

Ограничения адресного пространства

- Системные вызовы setrlimit/getrlimit
- Жесткий лимит (hard limit) — нельзя превышать
- Мягкий лимит (soft limit) — процесс может увеличивать и уменьшать
- RLIMIT_AS — лимит адресного пространства
- RLIMIT_STACK — лимит размера стека

Типы страниц в памяти

- Выгружаемые (страница может быть выгружена в область подкачки)
- Невыгружаемые (locked) — должны находиться в ОЗУ
- Процесс может пометить часть страниц как невыгружаемые (системный вызов mlock)
- Непривилегированный — макс. 32 KiB
- Все страницы ядра — невыгружаемые

Управление адресным пространством процесса

- Системный вызов brk() - изменить адрес конца сегмента данных

```
int brk(void *end_data_segment);
```

- Память в сегменте данных распределяется с помощью библиотечных вызовов malloc, calloc, realloc, free
- Библиотечные функции обычно не возвращают память ядру ОС даже после выполнения free

Стратегии распределения динамической памяти

- Битовый массив блоков
- Списки блоков

Битовый массив блоков

- Память разбивается на блоки выделения фиксированного размера (например, 16 байт)
- Каждому блоку выделения ставится в соответствие 1 бит в битовом массиве: 0 — блок свободен, 1 - занят

1	1	1	1	1	0	0	1	1	1	1	0	0	0

80 байт занято	32 св.	64 байта	48 байт

Списки блоков

- Пусть затребованный размер — N
- Реально выделяемый размер: ((max(8,N)+11)&~7)-4, то есть 12, 20, 28...
- 4 байта используются для поддержки списка блоков (хранят размер блока и бит занятости)

Алгоритмы выделения блоков

- Первый подходящий
- Самый подходящий
- Быстрый подходящий: поддерживаются список свободных блоков наиболее часто запрашиваемых размеров

Файлы, отображаемые в память

- Организация ввода-вывода через механизм страничной подкачки
- При необходимости подгрузки страницы в память (по прерыванию отсутствия страницы в памяти) страница подгружается из файла
- При необходимости выталкивания «грязной» страницы она записывается в файл

Системный вызов mmap

```
void *mmap(void *start, size_t length, int prot,
           int flags, int fd, off_t offset);
```

- start – желаемый адрес подключения к адресному пространству
- length – размер подключаемого блока памяти
- prot – флаги: PROT_EXEC, PROT_READ, PROT_WRITE
- fd – файловый дескриптор (-1 в некоторых случаях)
- offset – смещение в файле

Системный вызов mmap

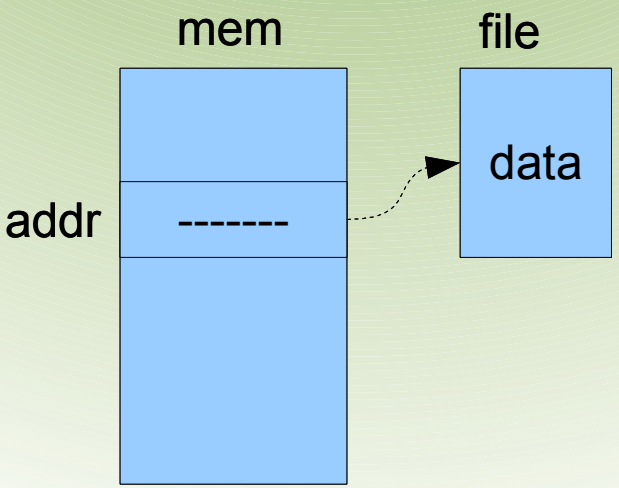
- flags: MAP_SHARED — разделяемое отображение, изменения в памяти отображаются обратно в файл
- MAP_PRIVATE — неразделяемое отображение, copy-on-write
- MAP_ANONYMOUS — анонимное отображение (не соответствует никакому файлу)
- MAP_FIXED — не пытаться размещать отображение по адресу, отличному от start
- MAP_NORESERVE — не резервировать область подкачки (для отображений, допускающих запись)

Copy-on-write

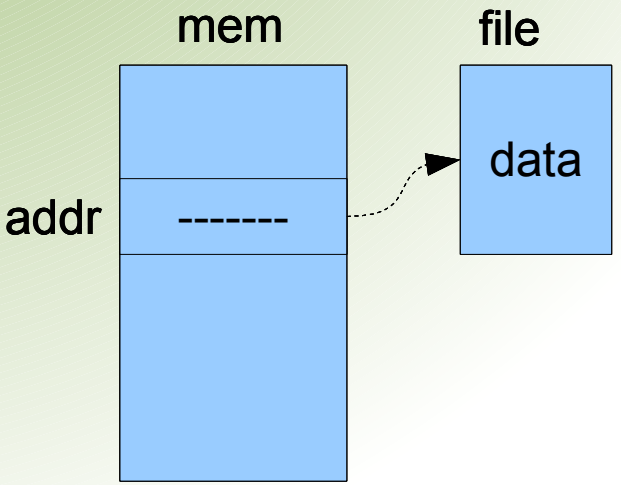
- Механизм оптимизации копирования страниц
- При обычном механизме копия страницы с выделением места в области подкачки создается немедленно
- При механизме copy-on-write создание копии страницы откладывается до первой записи в страницу

Copy-on-write

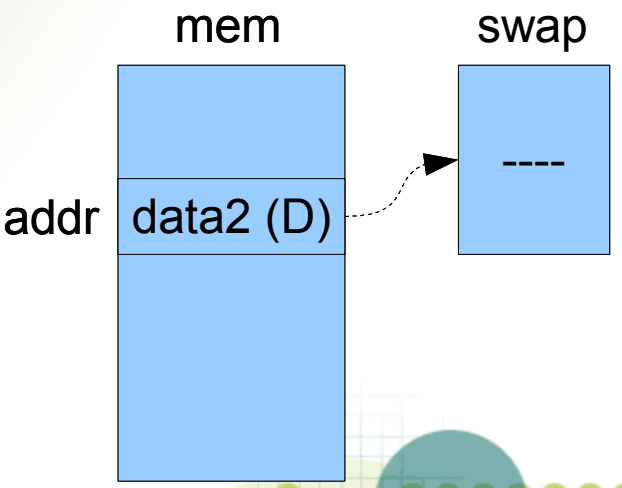
```
fd = open("file", O_RDWR, 0);
addr = mmap(0, 4096, PROT_READ|PROT_WRITE,MAP_PRIVATE,fd, 0);
```



При создании отображения страница в памяти помечена как отсутствующая, но отображенная на соответствующий файл



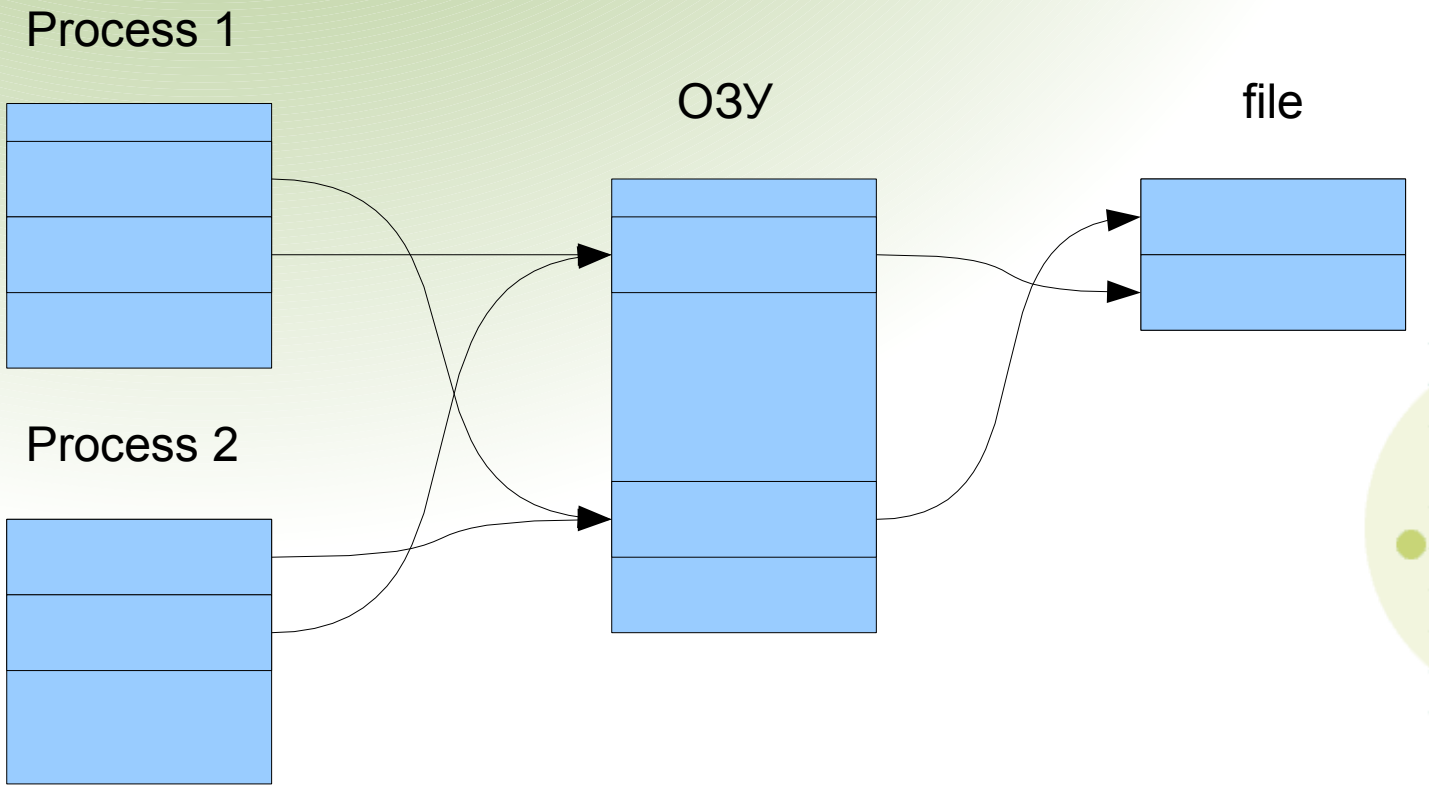
При чтении содержимое страницы подгружается из файла, страница помечается как «только для чтения»



При записи в страницу выделяется место в области подкачки, при необходимости создается копия страницы в ОЗУ, отображение переключается на swap

Разделение страниц между процессами

- Процессы, выполняющие отображение одного и того же файла, разделяют физические страницы ОЗУ



Необеспеченная память (memory overcommit)

- Стратегия выделения copy-on-write приводит к тому, что операция копирования страницы может быть выполнена в некоторый момент в будущем, причем прозрачным для выполняющегося процесса образом
- Когда потребуется создать копию страницы может оказаться, что память исчерпана и страница создана быть не может
- Необходимо снять с выполнения какой-нибудь процесс и таким образом освободить память (OOM killer)

Загрузка файла на выполнение

- Для загрузки файла используется mmap
- Страницы кода программы и данные только для чтения отображаются в режиме PROT_READ, MAP_PRIVATE
- Страницы данных программы отображаются в режиме PROT_WRITE, MAP_PRIVATE
- Таким образом код и данные только для чтения разделяются между процессами

Разделяемые библиотеки

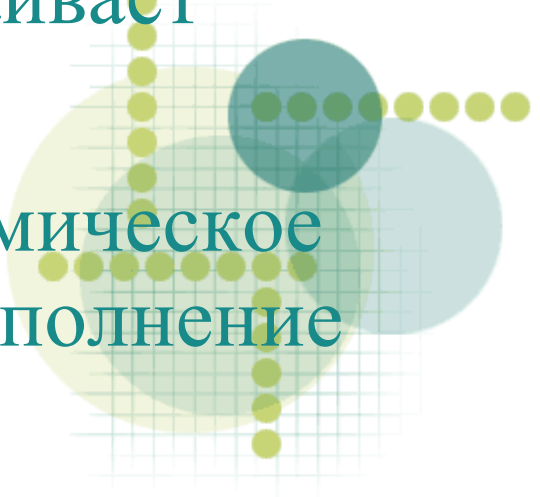
- Позволяют избежать дублирования кода в процессах (например, все процессы имеют общую реализацию printf)
- Делает возможным разделять код библиотек между процессами разных исполняемых файлов (при статической компоновке реализация printf может располагаться по разным адресам, что делает невозможным разделение)
- Облегчают обновление ПО

Разделяемые библиотеки

- Механизм загрузки разделяемых библиотек аналогичен механизму загрузки исполняемых файлов
- Динамические библиотеки могут загружаться в адресное пространство процессов по разным адресам
- Код разделяемых библиотек должен быть позиционно-независимым, чтобы избежать модификации сегмента кода

Позиционно-независимый код (PIC)

- Обращения к глобальным переменным и функциям выполняются косвенно через указатели в GOT (Global Offset Table)
- GOT позволяет сохранить секцию кода не изменяемой при перемещении библиотеки
- Динамический компоновщик настраивает только GOT
- С помощью GOT выполняется динамическое связывание имен при загрузке на выполнение
- Адрес GOT помещается в %ebx



Динамическое связывание

```
...
printf(...)
...
```

```
080482fc <printf@plt>:
 80482fc:        jmp    *0x804964c
 8048302:        push   $0x10
 8048307:        jmp    80482cc <_init+0x30>

 8048413:        call   80482fc <printf@plt>
```

```
GOT:
 …
 804964c:        .long  8048302
```

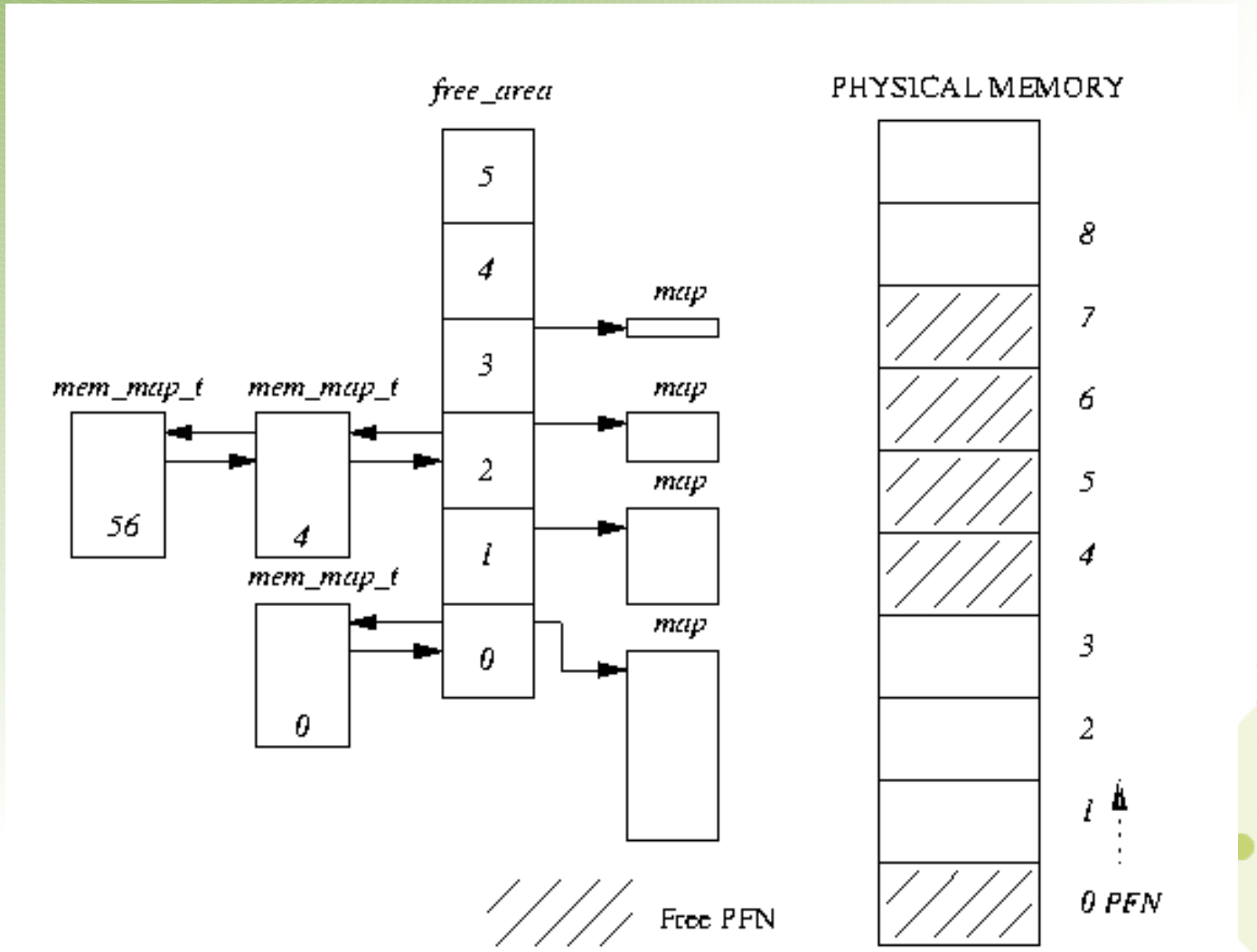
Управление памятью в ядре

- Выделение страниц
- Замещение страниц
- Буферный кеш блочных устройств

Учет физических страниц

- Каждой физической странице ставится в соответствие структура, хранящая информацию о странице
  - count — число пользователей данной странице (> 1 — страница используется несколькими процессами)
  - age - «возраст» страницы (для страничной подкачки)
  - map_nr — номер физической страницы

Выделение страниц (buddy)



Алгоритмы замещения страниц

- Для каждой страницы хранится два бита
  - R — из данной страницы было чтение
  - M — страница была модифицирована
- Бит R периодически (например, по таймерному прерыванию) очищается ядром ОС
- Бит M очищается только после записи страницы

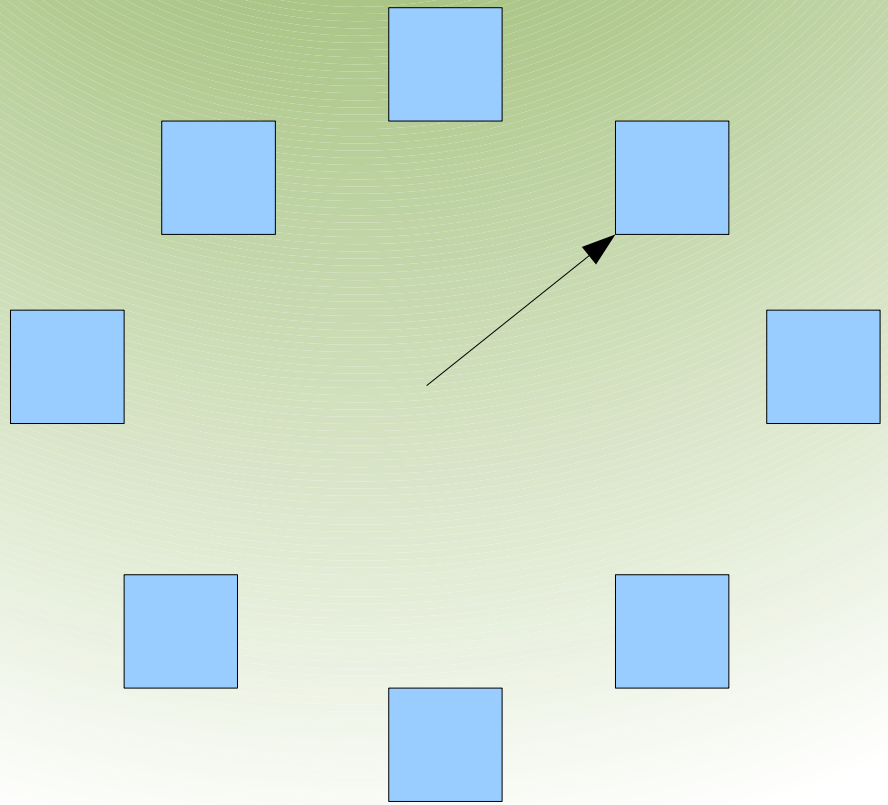
Алгоритм NRU

- Пытается удалять неиспользуемую в последнее время страницу (not recently used)
- Страница выбирается случайным образом из множества страниц наименьшего класса
  - R = 0, M = 0
  - R = 0, M = 1
  - R = 1, M = 0
  - R = 1, M = 1

Алгоритм FIFO

- Удаляется самая старая страница
- Алгоритм второй попытки: если к самой старой странице было обращение, время загрузки обновляется, страница переставляется в начало списка на удаление

Алгоритм часов



- Проверяется страница, на которую указывает стрелка
- R=0: страница выгружается
- R=1: R очищается, стрелка перемещается

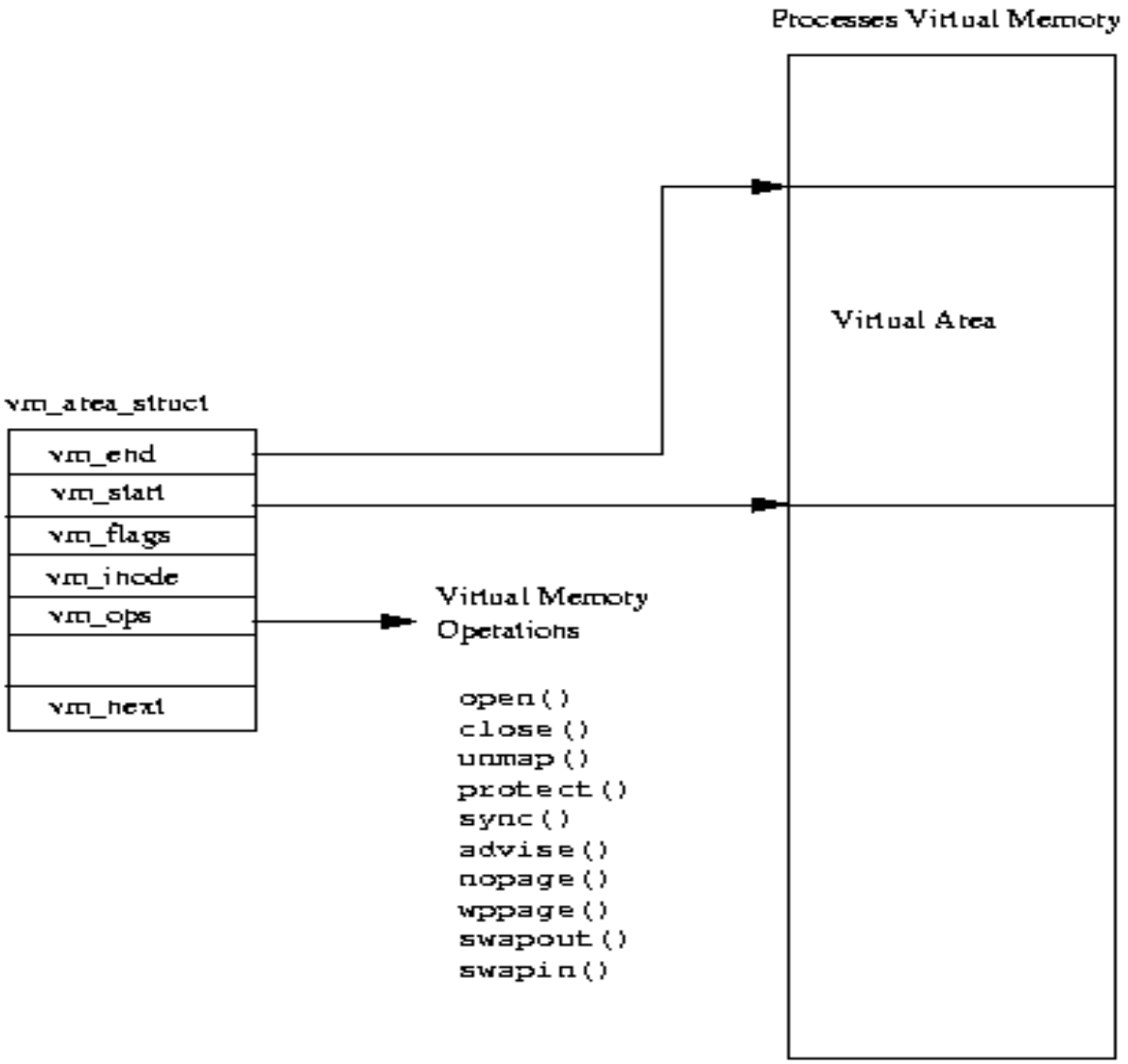
Алгоритм NFU (not frequently used)

- При каждом прерывании по таймеру к счетчику использования страницы прибавляется значение R
- Алгоритм старения: предположим, что под счетчик отводится K бит
- Значение счетчика для страницы пересчитывается по формуле:
  count = (count >> 1) | (1 << (K-1))

Страничный демон

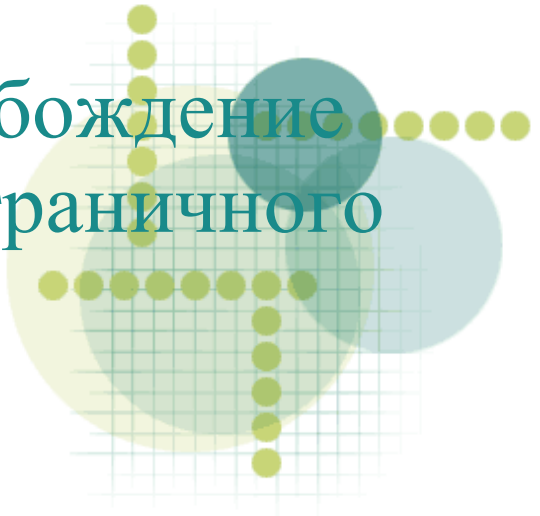
- В момент страничного прерывания запись «грязных» страниц нежелательна, т. к. еще более увеличивает время подкачки
- Страничный демон периодически проверяет наличие «грязных» страниц и сохраняет их
- В следующий раз при страничном прерывании страница будет «чистой»

Отображенные файлы



Буферный и страничный кеш

- Буферный кеш — кеширование работы с блок-ориентированным устройством.
- Страничный кеш — кеширует чтение из файла для файлов, отображенных в память (например, для исполняемых файлов и разделяемых библиотек)
- Страничный демон отвечает за освобождение «старых» страниц из буферного и страничного кеша



Страничный кеш