Лекция 10
Управление памятью

Ядро ОС и C++

- В основном, ядра операционных систем написаны на Си
- Symbian — практически полностью C++
- Windows, MacOS — драйвера можно разрабатывать на подмножестве C++

- Причины: 1) исторические — разработка началась в то время, когда C++ не было
- Объем кода — десятки млн. строк кода — затраты на перенос на C++

Ядро ОС и C++

- Технические причины: «Узкие места» C++
  - Исключения:
    - Либо накладные расходы на таблицы обработки исключений (раздувание кода)
    - Либо накладные расходы на поддержку стековых фреймов при работе (замедление работы)
    - «непрозрачные» передачи управления при работе
  - Динамическая память (STL):
    - Страницы памяти, занятые ядром, никогда не смогут использоваться в приложениях — требуется контроль за использованием памяти
  - «Непрозрачная» генерация кода

ООП в ядре

- Наследование (одиночное) — производная структура содержит структуру базового типа своим первым элементом
- Полиморфизм — структура указателей на функции (*_operations)

Управление памятью в ядре Linux

Дескриптор виртуальной памяти процесса

- struct mm_struct;
  - struct vm_area_struct *mmap;
    - Список областей памяти процесса
  - pgd_t *pgd;
    - Каталог виртуальных страниц верхнего уровня
  - atomic_t mm_users;
    - Счетчик использования из других процессов
  - atomic_t mm_count;
    - Основной счетчик использования

Дескриптор области виртуальной памяти

```
struct vm_area_struct {
    unsigned long vm_start;  /* Start address within */
    unsigned long vm_end;    /* The first byte after end */
    struct vm_area_struct *vm_next, *vm_prev;
    /* linked list of VM areas per task, sorted by address */
    struct mm_struct *vm_mm; /* The address space we belong*/
    pgprot_t vm_page_prot;   /* Access permissions of VMA*/
    unsigned long vm_flags;  /* Flags, see mm.h. */
    struct anon_vma *anon_vma;  /* анонимное отображение */
    const struct vm_operations_struct *vm_ops;
    unsigned long vm_pgoff;  /* Offset (within vm_file) */
    struct file * vm_file;   /* File map to (can be NULL)*/
    // …
};
```

Структура адресного пространства

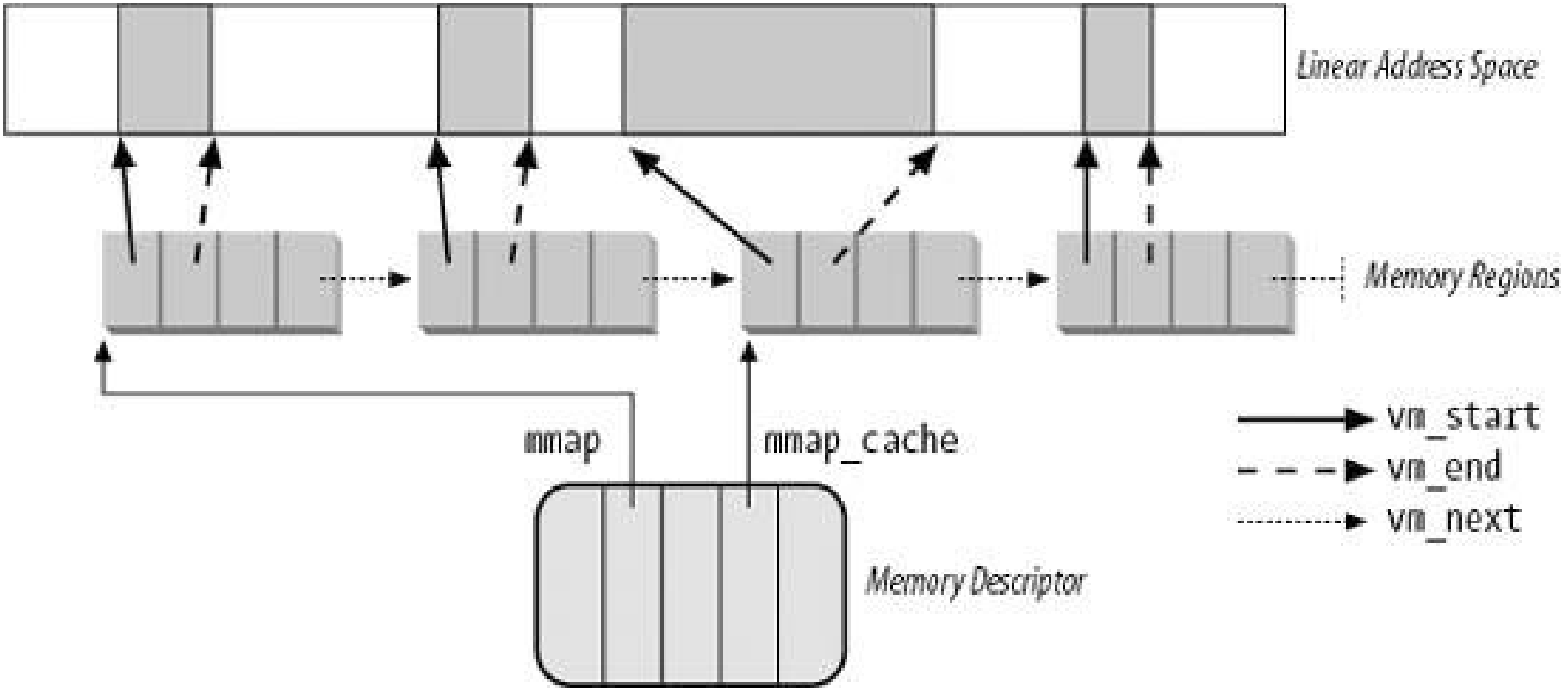
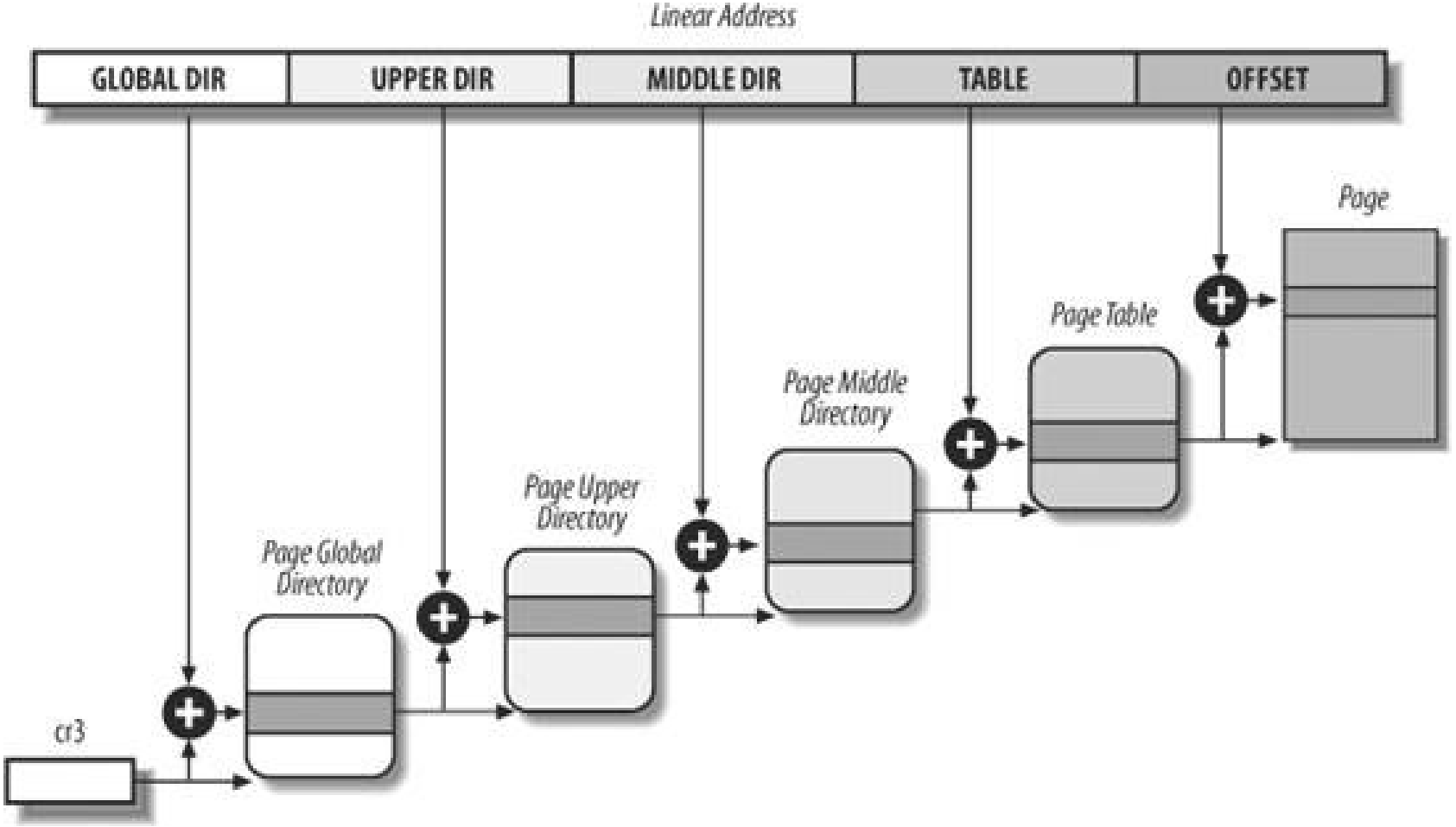


Таблица страниц в процессе

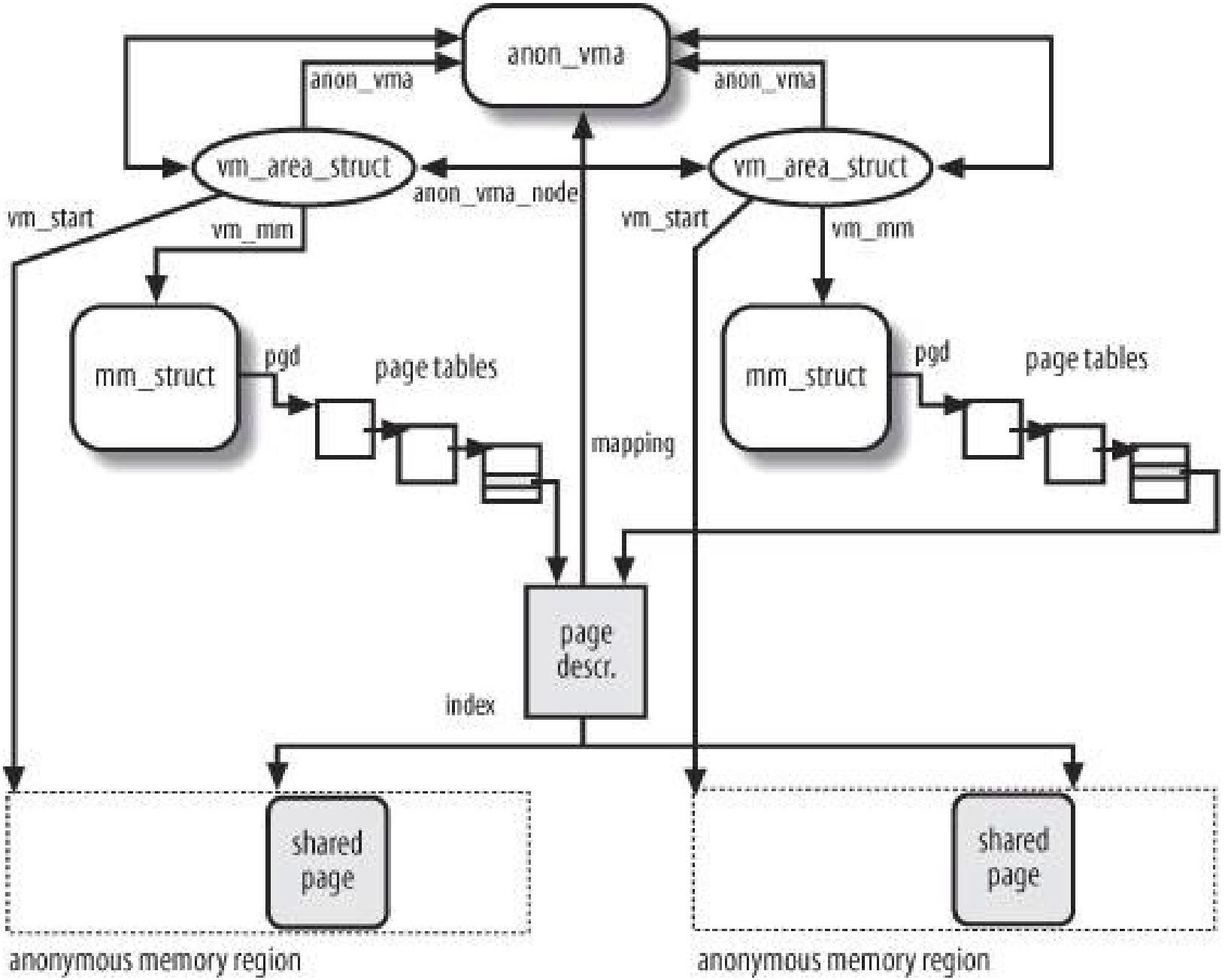
- Поле pgd в struct mm_struct
- Всегда четыре уровня страниц



Обратное отображение

- Задача: как внести изменения в структуры процессов при изменениях в состоянии физических страниц?
- У нас есть struct page, требуется найти все PTE процессов, которые ссылаются на эту физическую страницу
- Для анонимных отображений: anon_vma
- Для отображений из файла (mmap): struct address_space

anon_vma


anon_vma
anon_vma
anon_vma
vm_area_struct
anon_vma_node
vm_area_struct
vm_start
vm_mm
vm_start
vm_mm
mm_struct
pgd
page tables
mm_struct
pgd
page tables
mapping
page descr.
index
shared page
shared page
anonymous memory region
anonymous memory region

Struct address_space

```
struct address_space {
  struct inode            *host;     /* owning inode
*/
  struct radix_tree_root  page_tree; /* radix tree of
pages */
  struct prio_tree_root   i_mmap;    /* list of all
mappings*/
  unsigned long           nrpages;   /* total number
of pages */
  pgoff_t                 writeback_index;    /* WB
start off */
  struct address_space_operations   *a_ops;   /* op
table */
  struct backing_dev_info *backing_dev_info;  /* RA
info*/
};
```

Замещение страниц

- Алгоритм замещения страниц должен решить, какую из используемых страниц физической памяти освободить
- Может потребоваться запись в отображаемый файл или swap (для dirty страниц)
- Вызывается при Page Fault, когда нет свободных страниц или их меньше некоторого порога

Состояние страницы

- 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

Теоретически-оптимальный алг.

- (Алгоритм прорицателя :)
- Выгружается та страница памяти, которая не потребуется в будущем дольше всего
- Например, лучше выгрузить страницу, которая не будет нужна 5 секунд, чем страницу, которая не будет нужна 1 секунду
- Аккуратное предсказание в реальных условиях практически невозможно
- Можно собирать и накапливать информацию о предыдущих запусках, при последующих запусках поведение алгоритма будет приближаться к оптимальному

Not Recently Used

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

Алгоритм FIFO

- Страницы, загружаемые в память, добавляются в конец очереди
- Выгружается страница из начала очереди
- Алгоритм второй попытки: если к первой в очереди странице было обращение, время загрузки обновляется, страница переставляется в конец списка на удаление

Алгоритм часов (clock)

- Физические страницы организованы в кольцевой список
- «Стрелка» (итератор) указывает на некоторую страницу
- При Page Fault:
  - Если у текущей страницы R == 0, она замещается, стрелка продвигается
  - Если R == 1, R очищается, стрелка продвигается пока не найдется страница с R == 0

Least Recently Used (LRU)

- Выталкивается страница, которая не использовалась дольше всего
- Демонстрирует производительность, близкую к идеальной
- «Настоящий LRU»:
  - при каждом обращении к странице она переставляется в конец списка страниц на выталкивание
  - при выталкивании страница берется из начала
  - очень дорогой и практически не реализуемый!

Two-list LRU (Linux)

- Два списка: активный и неактивный
- Если у страницы R == 1 она перемещается в конец активного списка
- Если активный список становится слишком большим, страницы из начала активного списка перемещаются в конец неактивного списка
- Для вытеснения страницы берутся из головы неактивного списка

Not Frequently Used (NFU)

- 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))

Работа с SO-файлами

```
#include <dlfcn.h>


void *dlopen(const char *filename, int flag);
void *dlsym(void *handle, const char
*symbol);
int dlclose(void *handle);
```

gcc prog.c -o prog -ldl