Выравнивание (alignment) и дополнение (padding)

Выравнивание

- Выравнивание — гарантирует размещение переменной (простого или сложного типа) так, чтобы адрес размещения был кратен размеру выравнивания
- Дополнение — добавление в структуру скрытых полей так, чтобы поля структуры были правильно выровнены

Невыровненные данные

- Недопустимы на некоторых платформах (попытка обращения вызовет Bus Error)
- На других платформах (x86) обращение к невыровненным данным требует два цикла обращения к памяти вместо одного
- Работа с невыровненными данными **не атомарна**

Правильное выравнивание

- Тип char не требует выравнивания
- Short — выравнивание по двум байтам
- Int, long (x86), long long (x86), double (x86) — выравнивание по 4 байтам
- Long (x64), long long (x64), double (x64) — выравнивание по 8 байтам
- Выравнивание по границе 16 байтов — для линий кеша и для стека в Linux x86
- Выравнивание по границе 4096 — размер страницы (mmap)

Базовые типы и их свойства

type	X86 Linux		X64 Linux	
	size	alignment	size	alignment
`char`	1	1	1	1
`short`	2	2	2	2
`int`	4	4	4	4
`long`	4	4	8	8
`long long`	8	4	8	8
`void *`	4	4	8	8
`float`	4	4	4	4
`double`	8	4	8	8
`long double`	12	4	16	16

Пример:

```
struct s {
    char f1;
    long long f2;
    char f3;
};
```

- X86: sizeof(s) == 16
- X64: sizeof(s) == 24

```
struct s {
    long long f2;
    char f1;
    char f3;
};
```

- X86: sizeof(s) == 12
- X64: sizeof(s) == 16

Лекция 9
Управление памятью

Виртуальная адресация

- Для многопроцессной обработки требуется защита памяти: процесс не должен иметь неавторизованный доступ к памяти других процессов и ядра
- Адреса ячеек памяти данных и программы, используемые в процессе, не обязаны совпадать с адресами в физической памяти (ОЗУ)
- Адреса ячеек памяти для процесса — **виртуальные адреса**
- Адреса ячеек памяти в оперативной памяти — **физические адреса**

Виртуальная адресация (память)

- Программно-аппаратный механизм трансляции виртуальных адресов в физические
- Аппаратная часть — отображение виртуальных адресов в физические в «обычной» ситуации — должно быть очень быстрой, так как необходимо для выполнения каждой инструкции
- Программная часть — подготовка отображения к работе, обработка исключительных ситуаций

Модели виртуальной адресации

- Модель база+смещение
  - Два регистра для процесса: регистр базы (B), регистр размера (Z)
  - Пусть V — виртуальный адрес (беззнаковое значение), если V >= Z — ошибка доступа к памяти, иначе
  - P — физический адрес, P = B + V

Сегментная адресация

- Каждый процесс состоит из нескольких сегментов: сегмент кода, сегмент стека, сегмент данных1, сегмент данных2
- Для каждого сегмента хранятся свои базовый адрес и размер
- У каждого сегмента свои права доступа, например:
  - Код: чтение + выполнение
  - Стек: чтение + запись
- Сегмент может отсутствовать в оперативной памяти и подгружаться по требованию

Страничная адресация

- Все пространство виртуальных адресов разбивается на страницы **равного размера**
- Каждая страница виртуальной памяти отображается на физическую память независимо от других
- Каждая страница имеет права доступа независимо от других страниц
- Страница может быть отмечена как неотображенная или отсутствующая в памяти
- При невозможности аппаратно отобразить виртуальную страницу в физическую — Page Fault

Отображение страниц

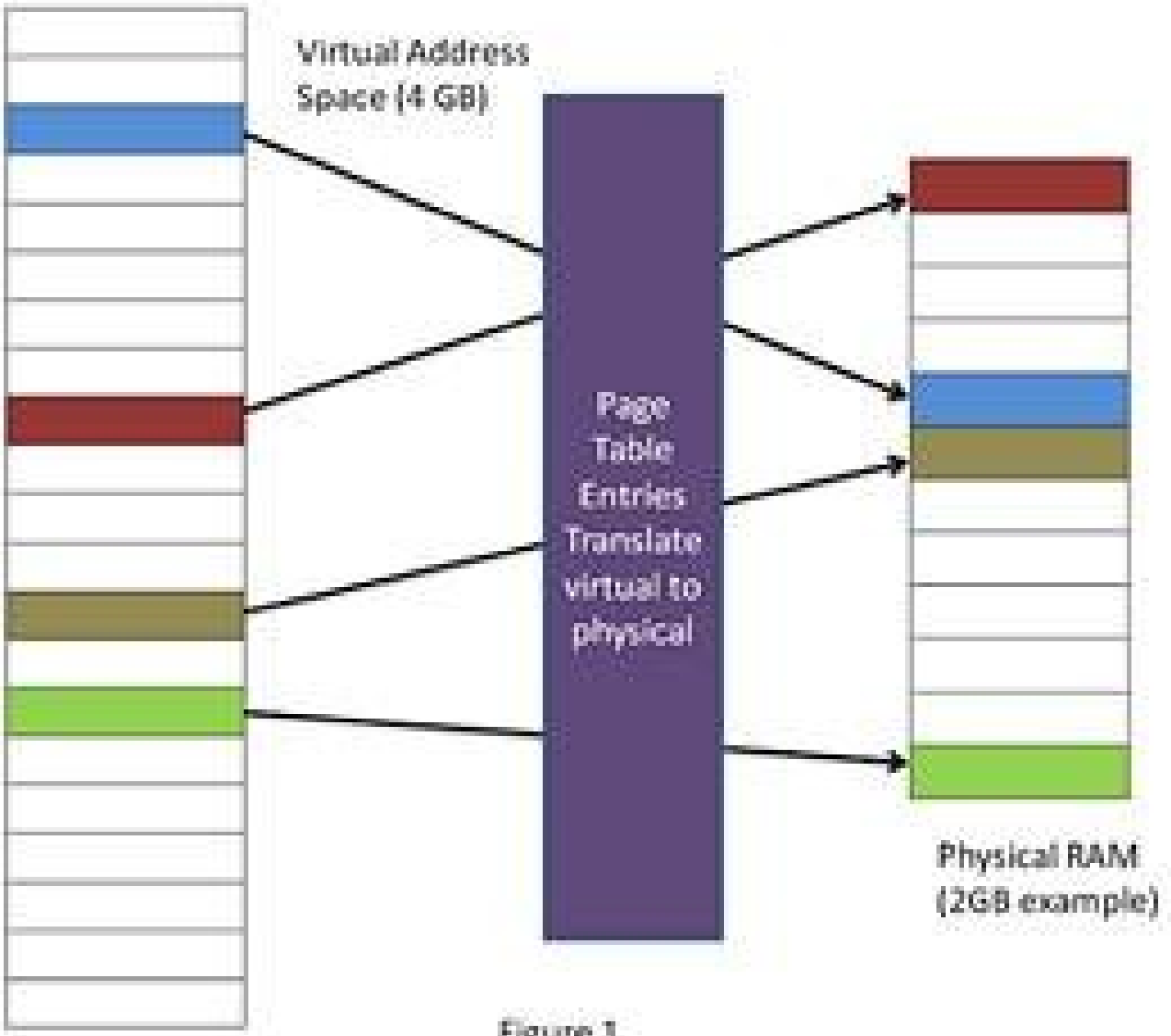
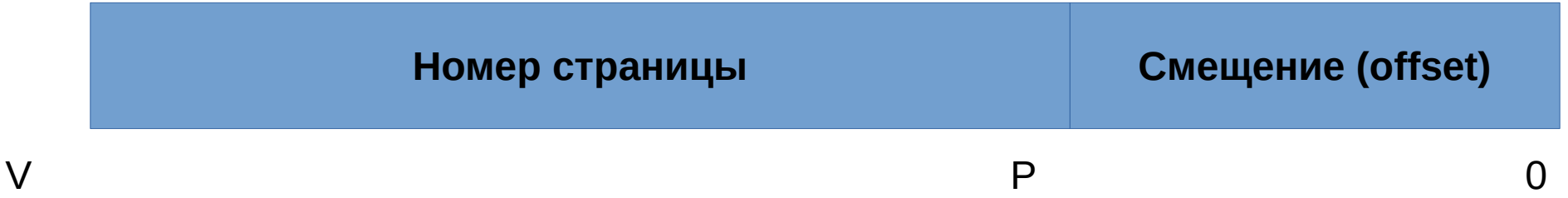


Figure 1

Виртуальный адрес



- V — количество бит виртуального адреса
- P — количество бит на смещение в страницу
- (V - P) — количество бит на номер страницы
- Для x86: V = 32, P = 12, V-P = 20
  - Виртуальное адресное пространство 4GiB
  - Размер страницы — 4096 байт (4KiB)
  - 2^20 (~1 Mi виртуальных страниц)

Двухуровневая таблица страниц (x86)

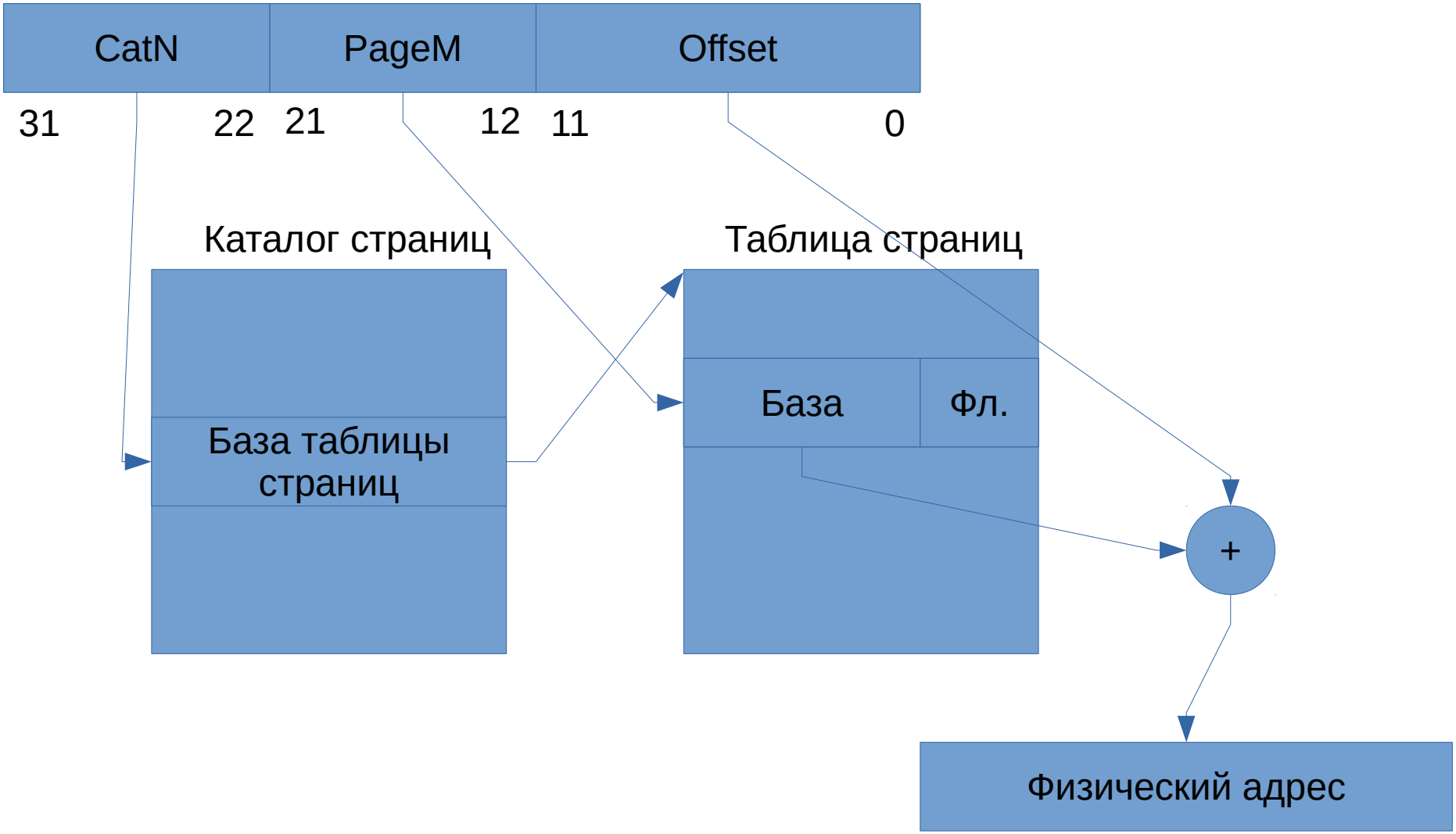


Таблица страниц

- Регистр процессора CR2 указывает на начало каталога страниц
- X86 — двухуровневая таблица страниц, размер страницы — 4KiB, в каталоге страниц 1024 записи, в каждой таблице страниц 1024 записи, одна запись — 4 байта
- X64 — четырехуровневая таблица страниц, размер страницы — 4KiB, в таблице каждого уровня 512 записей, одна запись — 8 байт.

Элемент таблицы страниц (x86)

- P — страница присутствует в ОЗУ
- R — право на запись в страницу
- U — доступна из user-space
- C — кеширование страницы запрещено
- W — разрешена сквозная (write-through) запись
- A — к странице было обращение
- D — (dirty) страница была модифицирована
- G — страница глобальная

Трансляция адресов x86

```
#define PAGE_SIZE  4096
#define TABLE_SIZE 1024
unsigned translate(unsigned va)
{
  unsigned *catalog = CR2;
  unsigned *table = catalog[va >> 22] & -PAGE_SIZE;
  unsigned phys = table[(va >> 12) & (TABLE_SIZE - 1)]
& -PAGE_SIZE;
  return phys + (va & (PAGE_SIZE - 1));
}
```

Доступ к странице

- Если страница отсутствует в ОЗУ (P == 0), обращение к странице → Page Fault
- Если в user-space и U == 0 → Page Fault
- Если записываем в страницу и R == 0 → Page Fault
- Устанавливаем флаг «accessed» (A = 1)
- Если записываем, устанавливаем флаг «dirty» (D = 1)

TLB (Translation Lookaside Buffer)

- Двухуровневая таблица страниц может потребовать 2 вспомогательных обращения к памяти!
- TLB — кэш-память для отображения виртуального адреса в физический
- TLB может быть многоуровневым и разделенным: для Intel Nehalem:
  - 64 записи в L1 DTLB
  - 128 записей в L1 ITLB
  - 512 записей в L2 TLB

PageFault

- Исключение PageFault не обязательно ошибка в программе
- Допустимые ситуации:
  - Страница данных откачана в swap (P == 0)
  - Страница кода не загружена из файла (P == 0)
  - Запись в страницу созданную для copy-on-write (R == 0)
- Обработчик исключения определяет причину PageFault. Если PageFault произошел из-за ошибки, ошибка передается в программу

3G/1G virtual memory split

- Верхний 1GiB адресного пространства процесса содержит страницы, отображенные в режиме U == 0
- Страницы недоступны из user-space, но как только процесс переключается в kernel-space они становятся доступны
- Код и данные ядра отображаются в эту часть адресного пространства
- При переключении контекста (при входе в системный вызов) нет накладных расходов на перезагрузку таблицы страниц
- Из кода ядра непосредственно доступна память процесса (copy_from_user, copy_to_user)

Управление памятью в ядре Linux

Управление физической памятью

- struct page; - дескриптор физической страницы — создается для каждой страницы
- sizeof(struct page) == 32 на x86
- struct page *mem_map; - массив всех физических страниц
- Если размер ОЗУ 4GiB, то массив mem_map займет 32MiB пространства ядра
- Определена в include/linux/mm_types.h

Struct page

- unsigned long flags;
  - PG_locked; - не может быть выгружена
  - PG_error; - I/O error на этой странице
  - PG_referenced; - было обращение к странице
  - PG_uptodate; - содержимое корректно
  - PG_dirty; - требует записи на диск
- struct address_space *mapping;
  - Либо указатель на пространство inode,
  - Либо на anon_vma (для анонимных страниц)

Struct page

- atomic_t _mapcount;
  - Счетчик использований данной физической страницы в PTE (page table entry) виртуального адресного пространства страниц
- atomic_t _count;
  - Счетчик ссылок
- struct list_head lru;
  - Поля prev, next, которые организуют страницы в список LRU
- pgoff_t index;
  - Смещение в отображении mmap
- struct address_space *mapping;
  - Структура, описывающее отображение страницы (inode)

Зоны физической памяти

- Области физической памяти
  - ZONE_DMA — пригодные для DMA
  - ZONE_NORMAL — непосредственно отображенные в виртуальное адресное пространство ядра
  - ZONE_HIGH — динамически отображаемые в виртуальное адресное пространство ядра
- struct zone; - структура, описывающая зоны
  - zone_lru — голова LRU списка страниц зоны
- struct zone *node_zones; - массив зон памяти