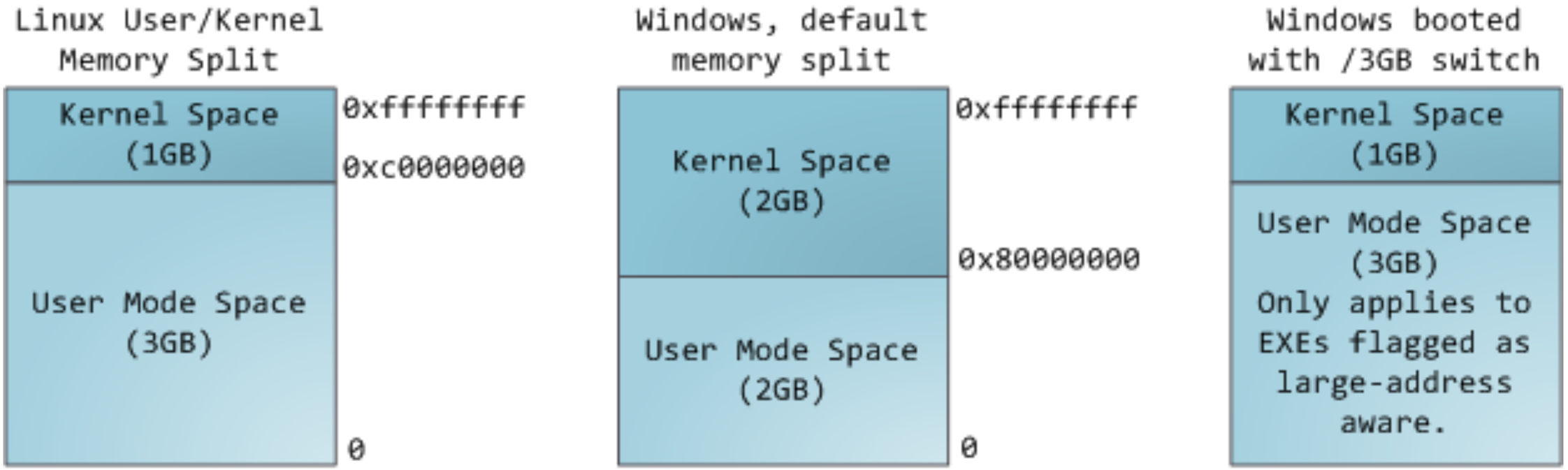
Операционные системы и сети

Управление памятью

Павел Филонов

Распределение памяти

Для x86

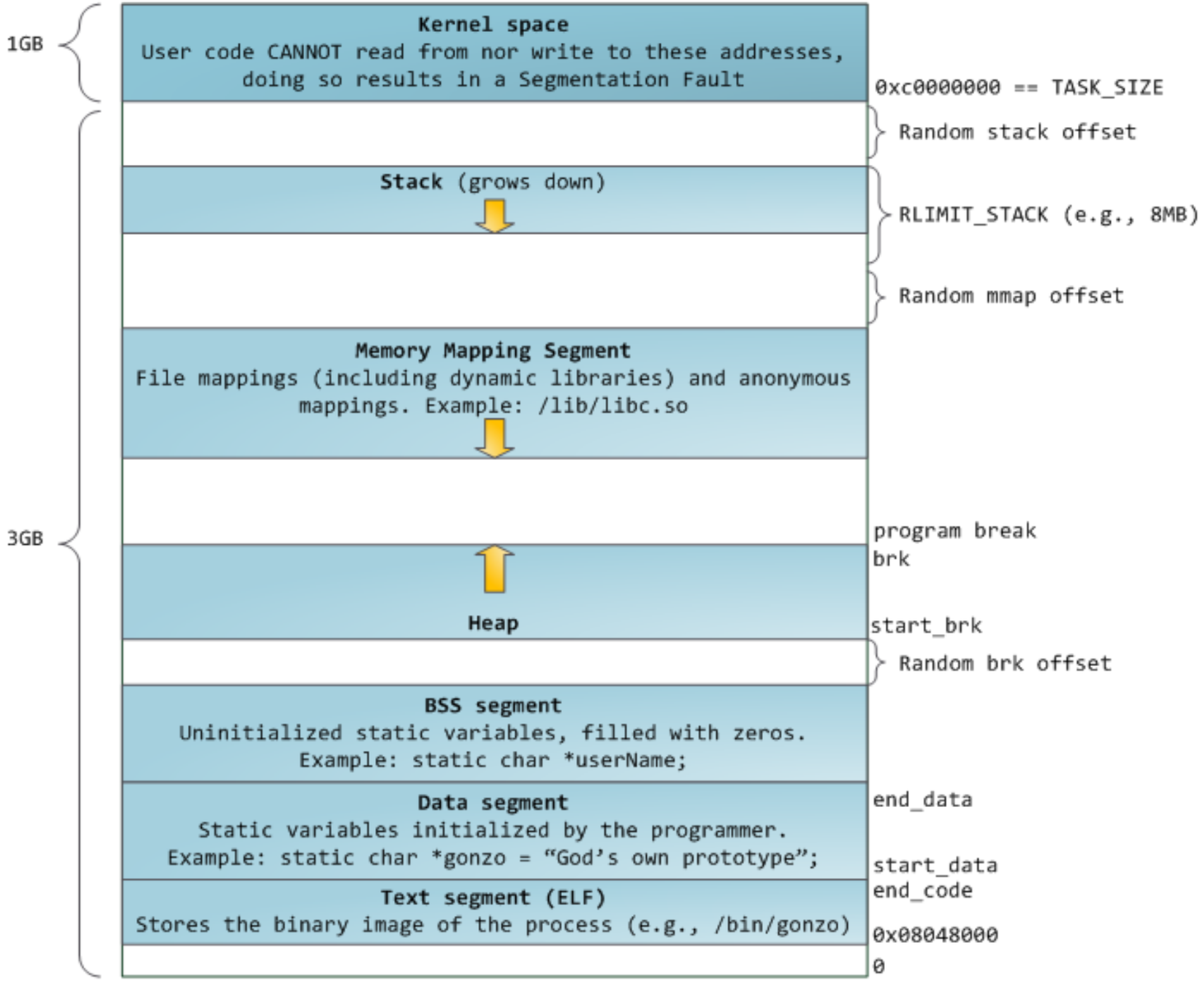


Для x86-64

Все поровну. И ядру и пользовательскому процессу по 128 Tb

Распределение памяти процесса

Linux x86



Пример. Сегменты разных процессов

cat /proc/self/maps | awk '{print $1,$2,$3,$6}'

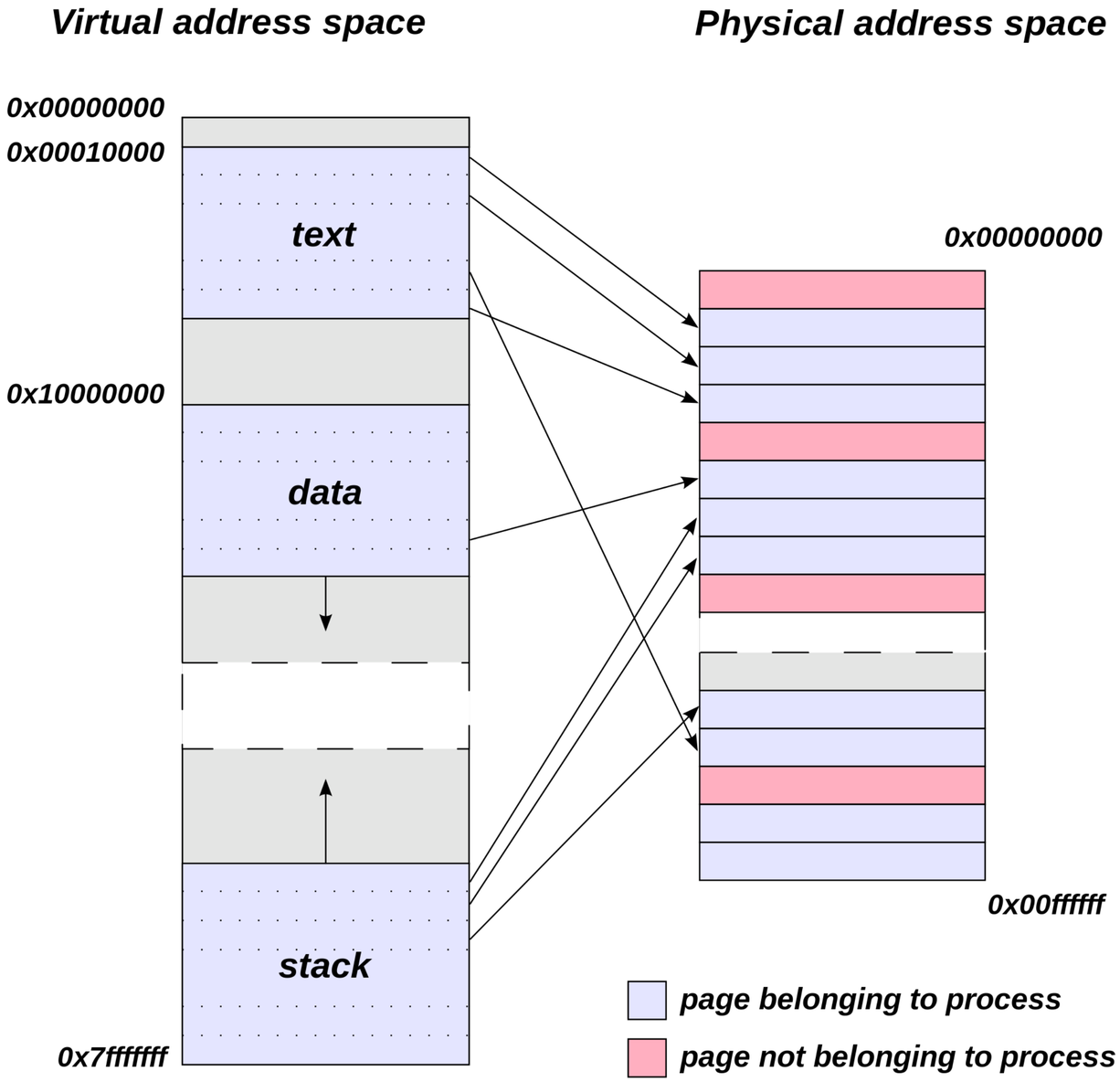
```
address                   perms offset   path
55e680665000-55e68066d000 r-xp  00000000 /bin/cat  // .text
55e68086c000-55e68086d000 r--p  00007000 /bin/cat
55e68086d000-55e68086e000 rw-p  00008000 /bin/cat  // .data
55e681067000-55e681088000 rw-p  00000000 [heap]
7f5bc079d000-7f5bc0984000 r-xp  00000000 /lib/x86_64-linux-gnu/libc-2.27.so
7f5bc0984000-7f5bc0b84000 ---p  001e7000 /lib/x86_64-linux-gnu/libc-2.27.so
7f5bc0b84000-7f5bc0b88000 r--p  001e7000 /lib/x86_64-linux-gnu/libc-2.27.so
7f5bc0b88000-7f5bc0b8a000 rw-p  001eb000 /lib/x86_64-linux-gnu/libc-2.27.so
7f5bc0b8a000-7f5bc0b8e000 rw-p  00000000
7f5bc0b8e000-7f5bc0bb5000 r-xp  00000000 /lib/x86_64-linux-gnu/ld-2.27.so
7f5bc0d8a000-7f5bc0dae000 rw-p  00000000
7f5bc0db5000-7f5bc0db6000 r--p  00027000 /lib/x86_64-linux-gnu/ld-2.27.so
7f5bc0db6000-7f5bc0db7000 rw-p  00028000 /lib/x86_64-linux-gnu/ld-2.27.so
7f5bc0db7000-7f5bc0db8000 rw-p  00000000
7ffc16ef9000-7ffc16f1a000 rw-p  00000000 [stack]
7ffc16fe4000-7ffc16fe6000 r--p  00000000 [vvar]
7ffc16fe6000-7ffc16fe8000 r-xp  00000000 [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 [vsyscall]
```

Задачи управления памятью

- Защита оперативной памяти
- Уменьшение дублирования данных
- Перемещение кода
- Фрагментация
- Подкачка (swapping)

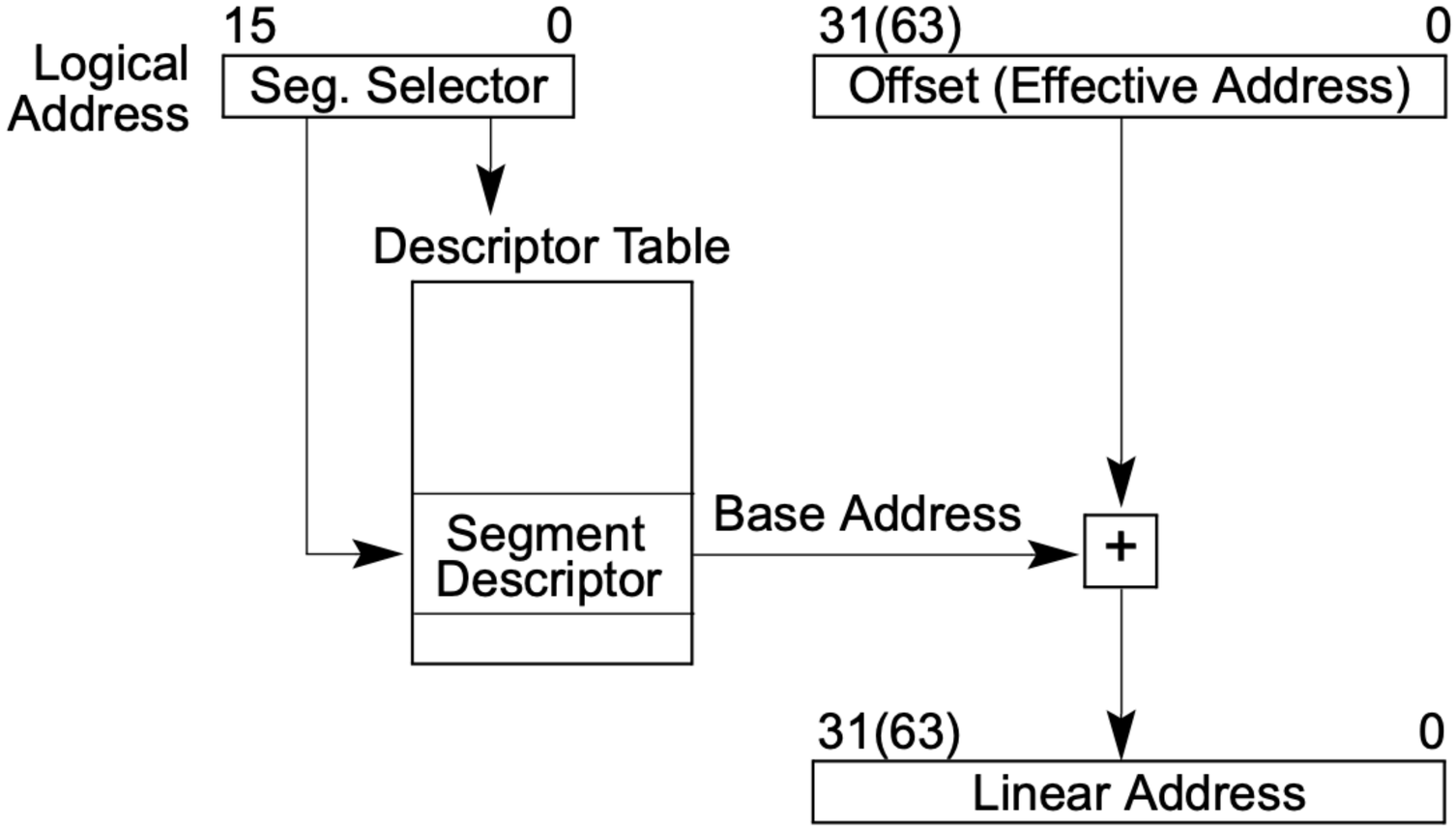
Виртуальная память

- **Виртуальная память** - способ организации памяти, при котором адреса, используемые в машинных командах, являются абстрактными (виртуальными) и отображаются на другие адреса физической (реальной памяти).



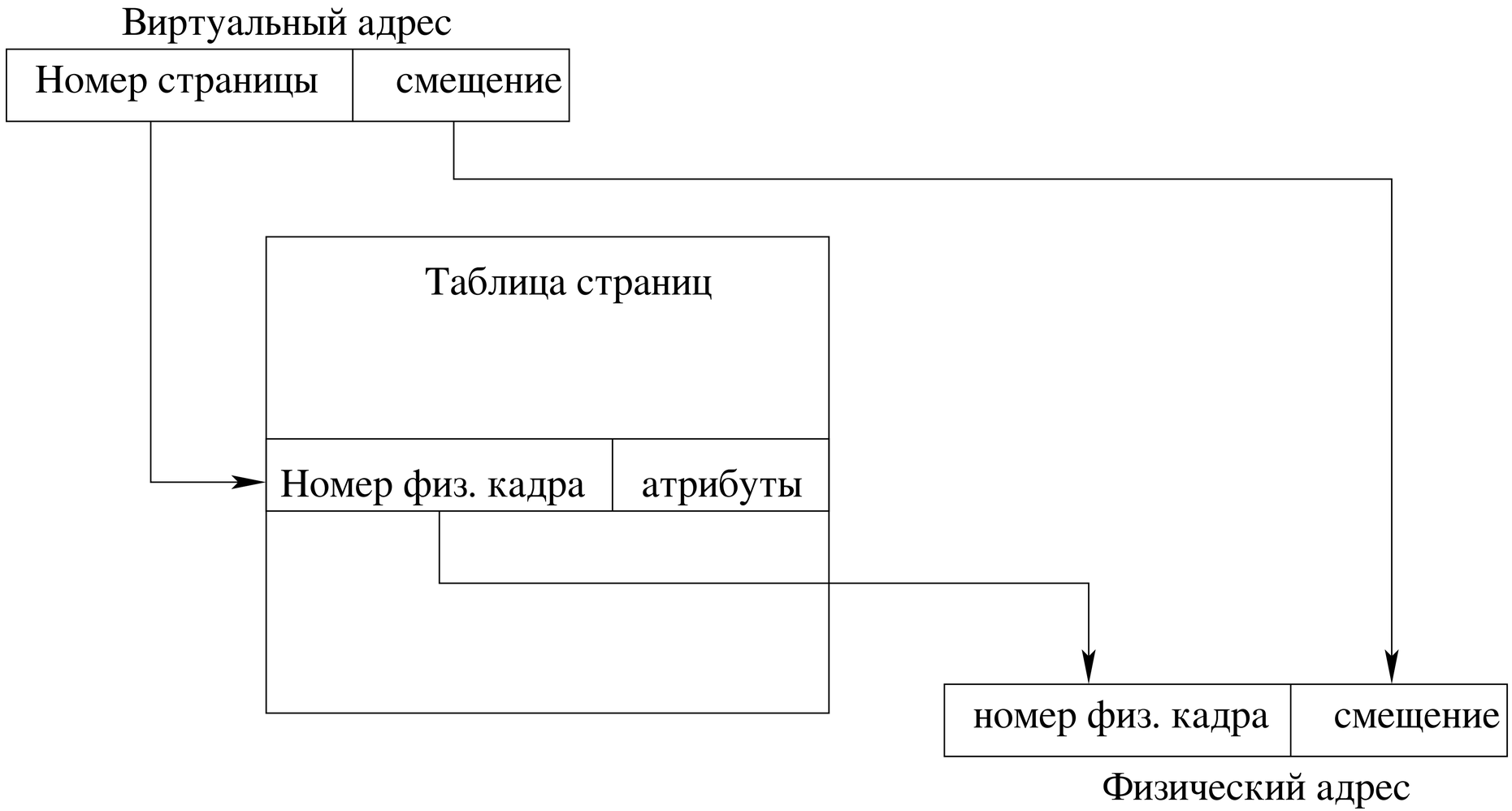
Сегментная модель памяти

Адреса сегментов хранятся в специальных сегментных регистрах CS, DS, SS, ES, FS, GS



Страничная модель памяти

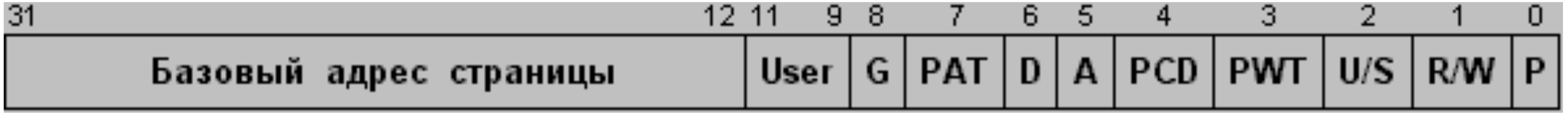
- Страница памяти имеет фиксированный размер. Например 4 Кб



Структура таблицы страниц

На примере архитектуры x86

Таблица страниц (Page Table) состоит из 4-байтовых элементов (Entries). Эти элементы называются PTE (Page Table Entries) и представляют собой по сути - указатели на страницы, по формату - структуры данных.



Структура таблицы страниц

Атрибуты страницы

1. P (Present - присутствие). Если 0, то страница не отображена на физическую память.
2. R / W (Read / Write - Чтение / Запись). Если 0, то для этой страницы разрешено только чтение, 1 - чтение и запись.
3. U / S (User / Supervisor - Пользователь / Система). Если 0, то доступ к странице разрешён только с нулевого уровня привилегий, если 1 - то со всех.
4. PWT (Write-Through - Сквозная запись). Когда этот флаг установлен, разрешено кэширование сквозной записи (write-through) для данной страницы, когда сброшен - кэширование обратной записи (write-back).
5. PCD (Cache Disabled - Кэширование запрещено).
6. A (Accessed - Доступ). Устанавливается процессором каждый раз, когда он производит обращение к данной странице.
7. D (Dirty - Грязный). Устанавливается каждый раз, когда процессор производит запись в данную страницу.
8. PAT (Page Table Attribute Index - Индекс атрибута таблицы страниц). Для процессоров, которые используют таблицу атрибутов страниц (PAT - page attribute table).
9. G (Global Page - Глобальная страница). Когда установлен, определяет глобальную страницу.
10. Биты с 9 по 11 не используются процессором.
11. Биты с 12 по 31 несут в себе базовый адрес страницы, с которого начинается страница.

Если страница не присутствует в памяти (бит P=0), то процессор не использует все остальные биты элемента PTE и программа может их использовать по своему усмотрению.

Недостатки одноуровневой таблицы

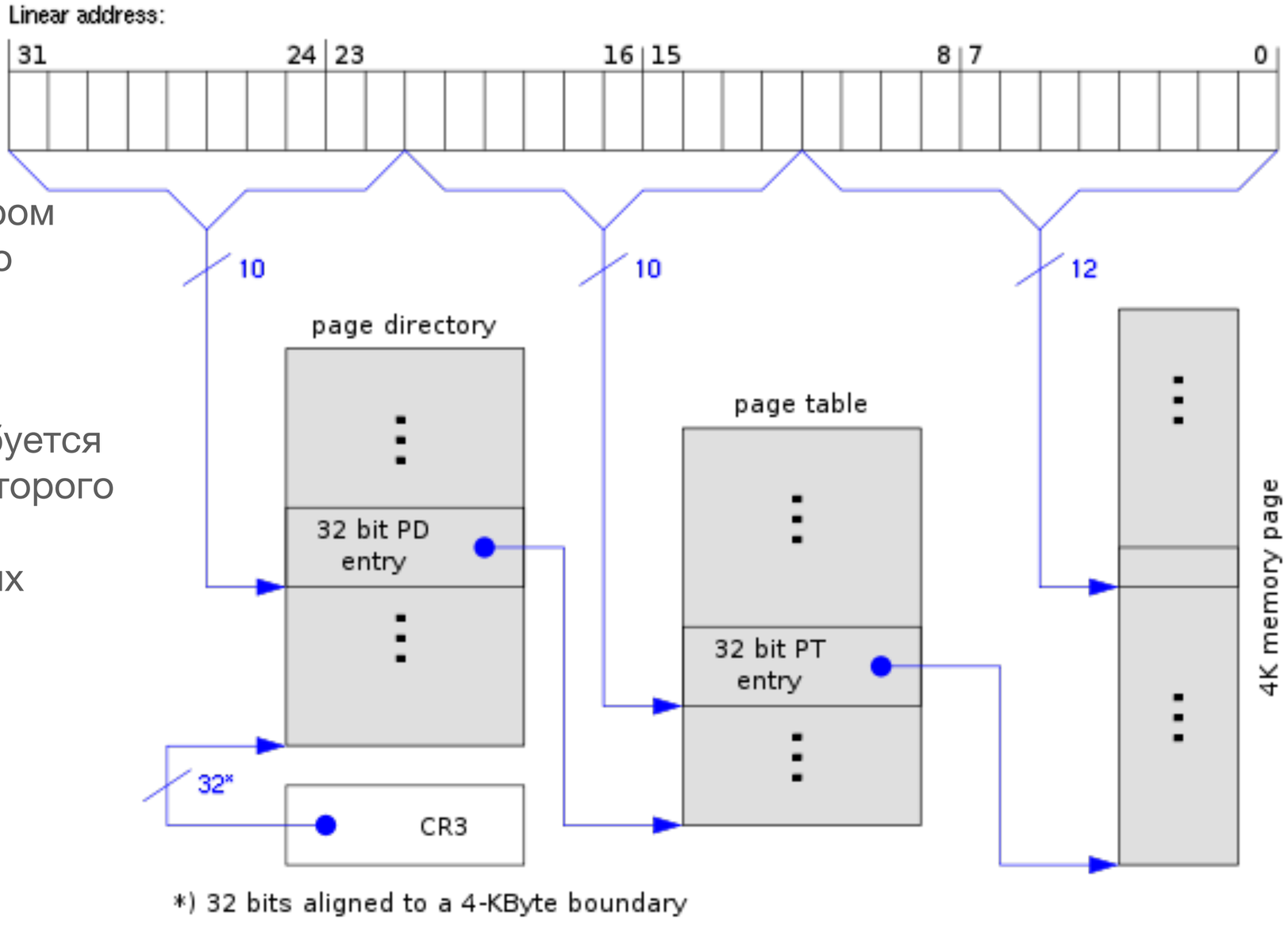
На примере архитектуры x86

- Пусть размер страницы PS = 4 Кб = 2**12 байт
- Всего процесс может адресовать 4 Гб = 2**32 байт
- Каждая запись в таблице страниц занимает 4 байта = 2**2 байт
- Всего нужно 4 Гб / 4 Кб = 2**32/2**12 = 2**20 строк в таблице
- Размер такой таблицы 2**20 * 2**2 байт = 4 Мб
- Для каждого процесса нужна своя таблица страниц
- Итого 4 Мб накладных расходов для каждого процесса

Двухуровневая таблица страниц

CR3 - регистр процессора, в котором хранится адрес таблицы первого уровня

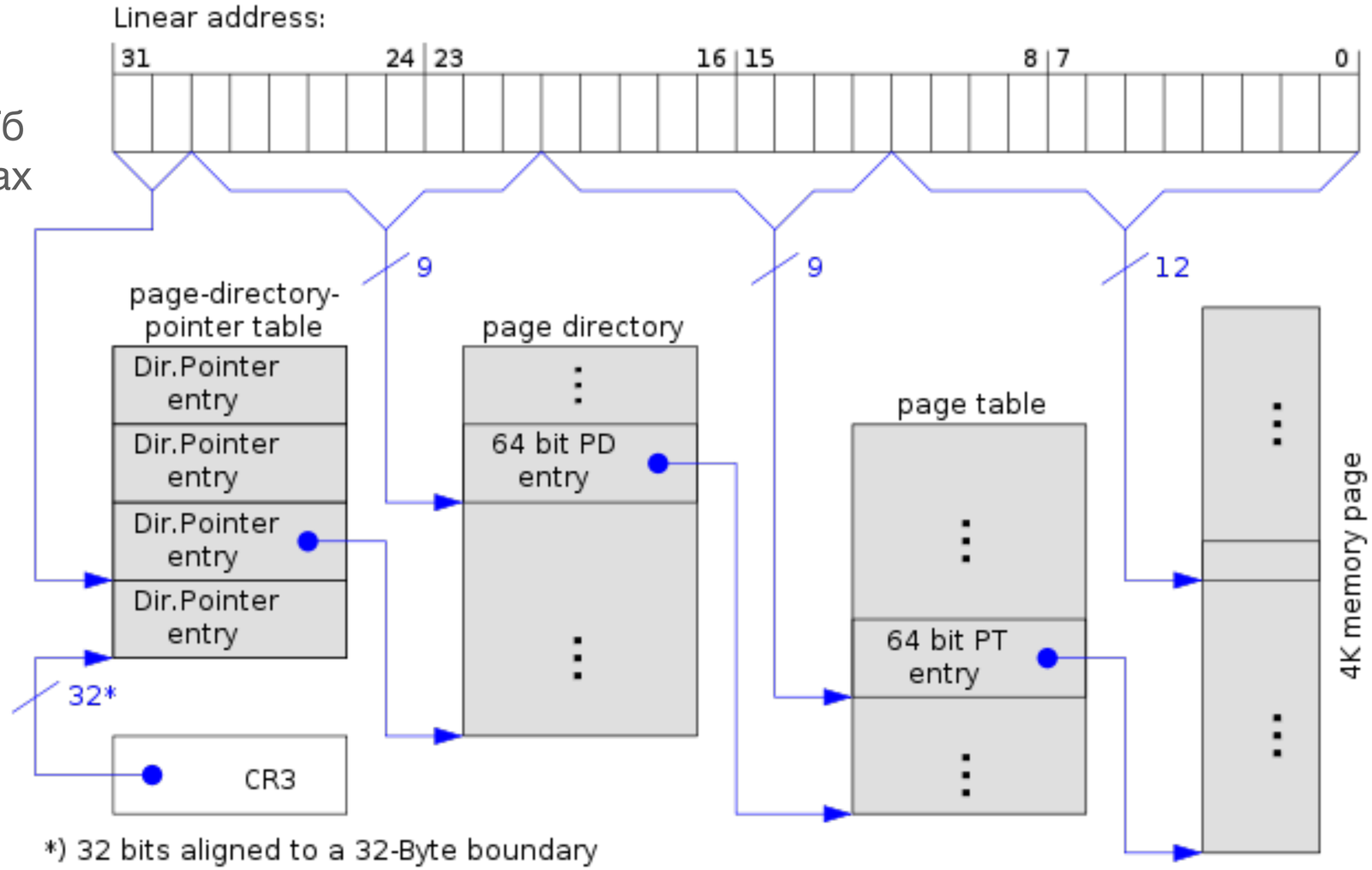
При 2-хуровневой организации требуется минимум 2 таблицы (1 первого и 1 второго уровней)
Минимум получается **8 Кб** накладных расходов (обычно больше)



Трехуровневая страничная модель

Технология PAE (Physical Address Extension) x86

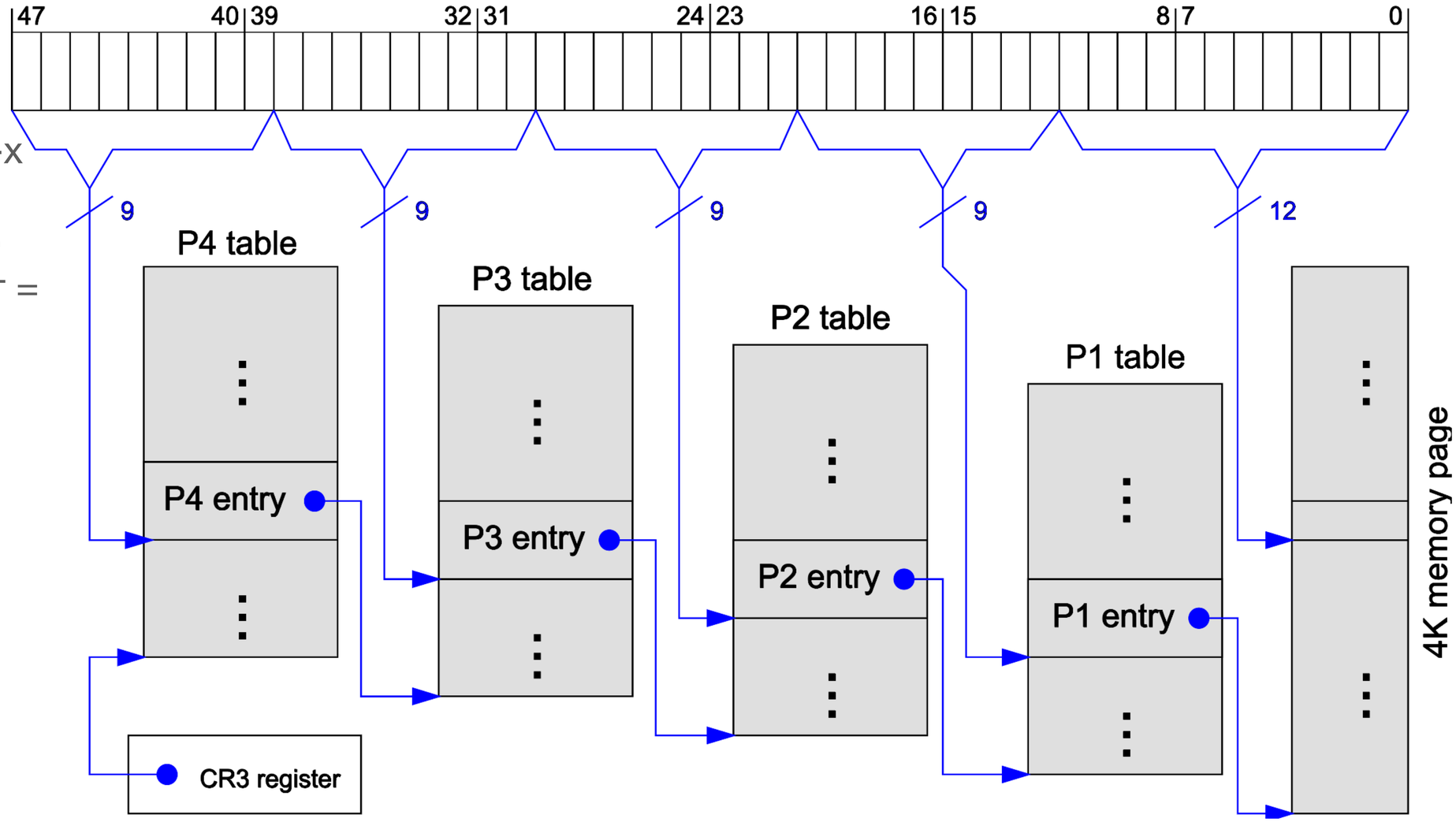
Позволяет адресовать больше 4 Гб памяти при 32-х разрядных адресах



4-хуровневая страничка модель

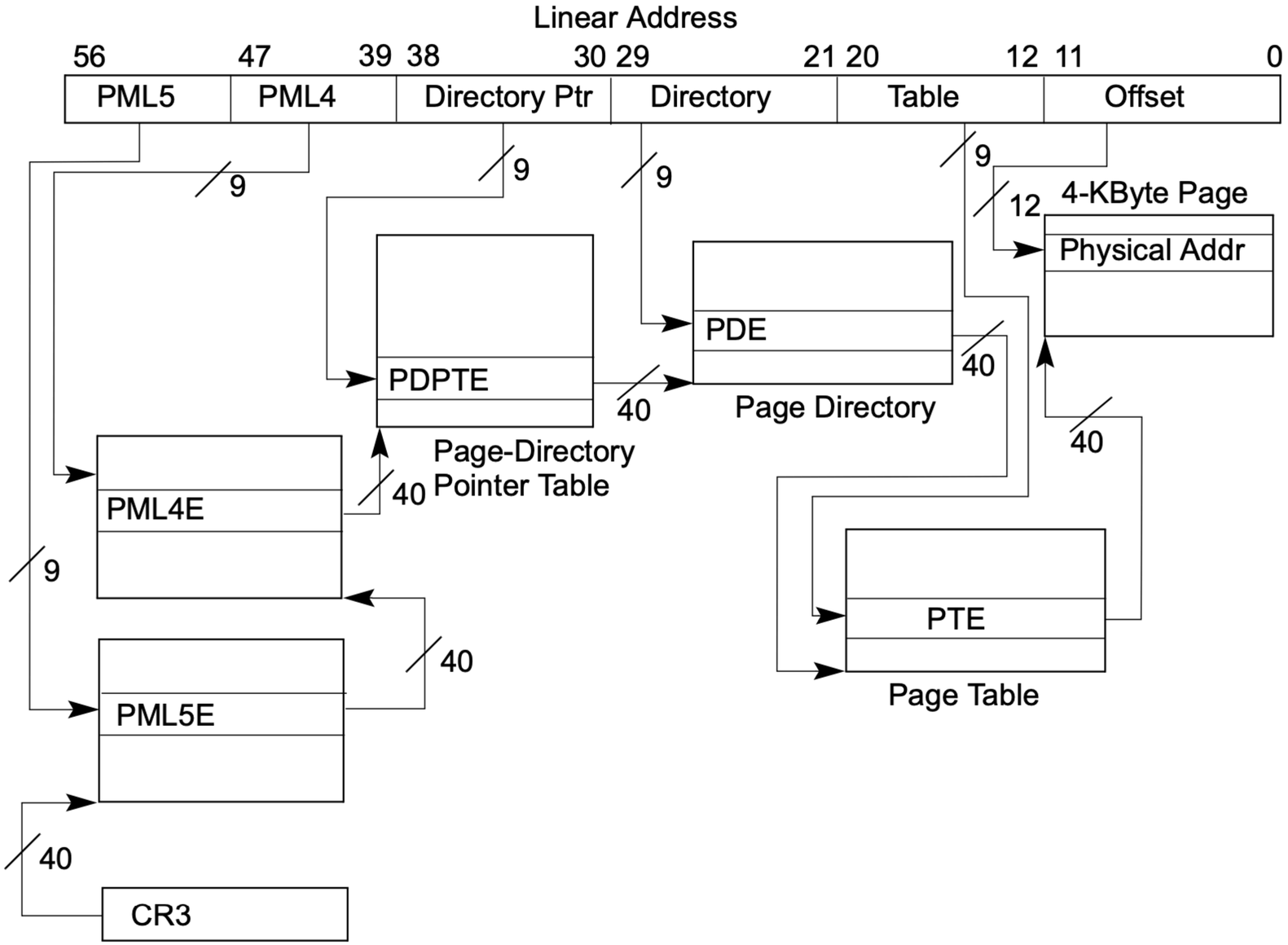
Архитектура x86-64

В x86-64 используются 64-х битные регистры, но для адресации памяти можно использовать только 48 бит = 256 Тб



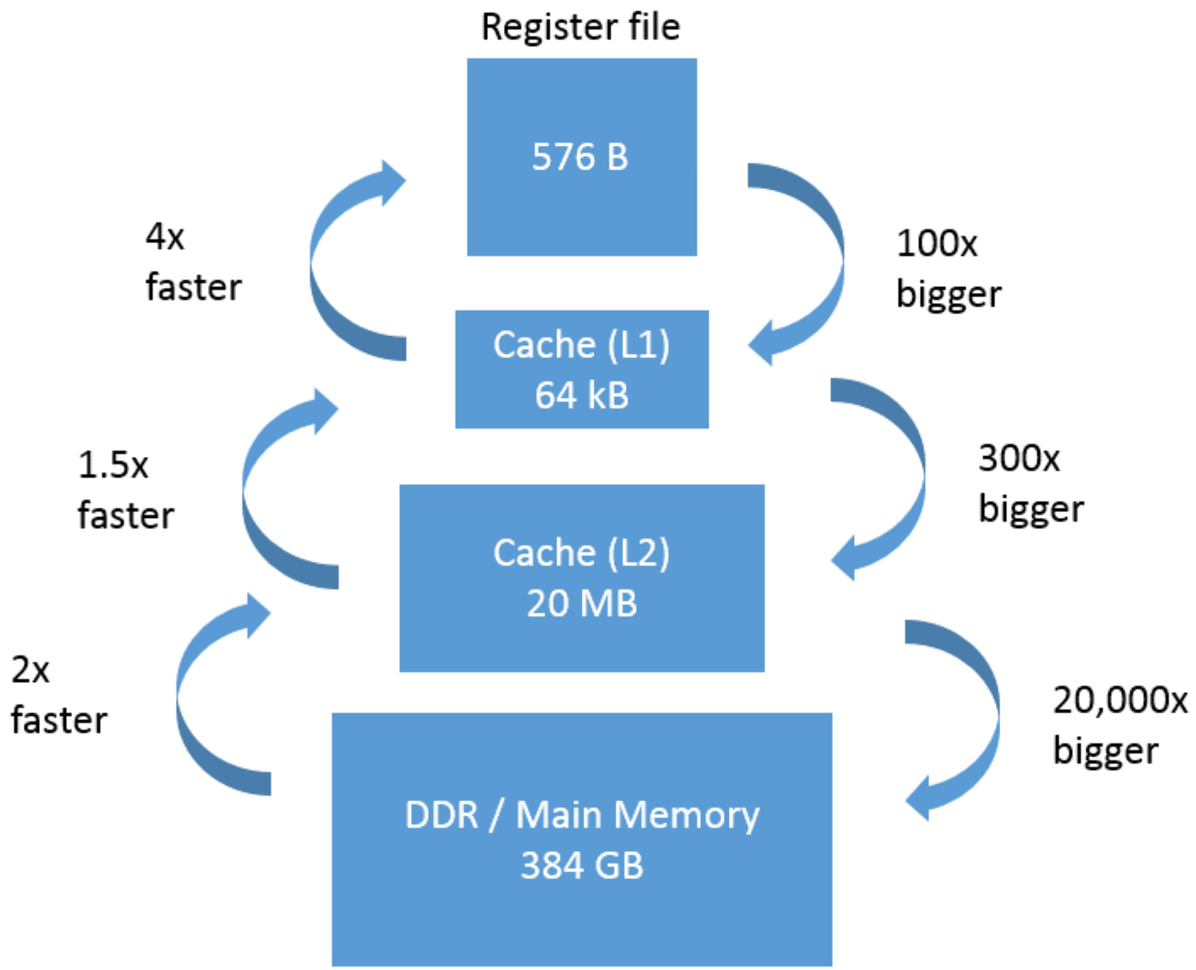
5-уровневая страничная модель

Расширение x86-64, которое позволяет использовать 57 бит для адресации памяти = 128 Пб



Бесплатных завтраков не бывает

Иерархия памяти процессора



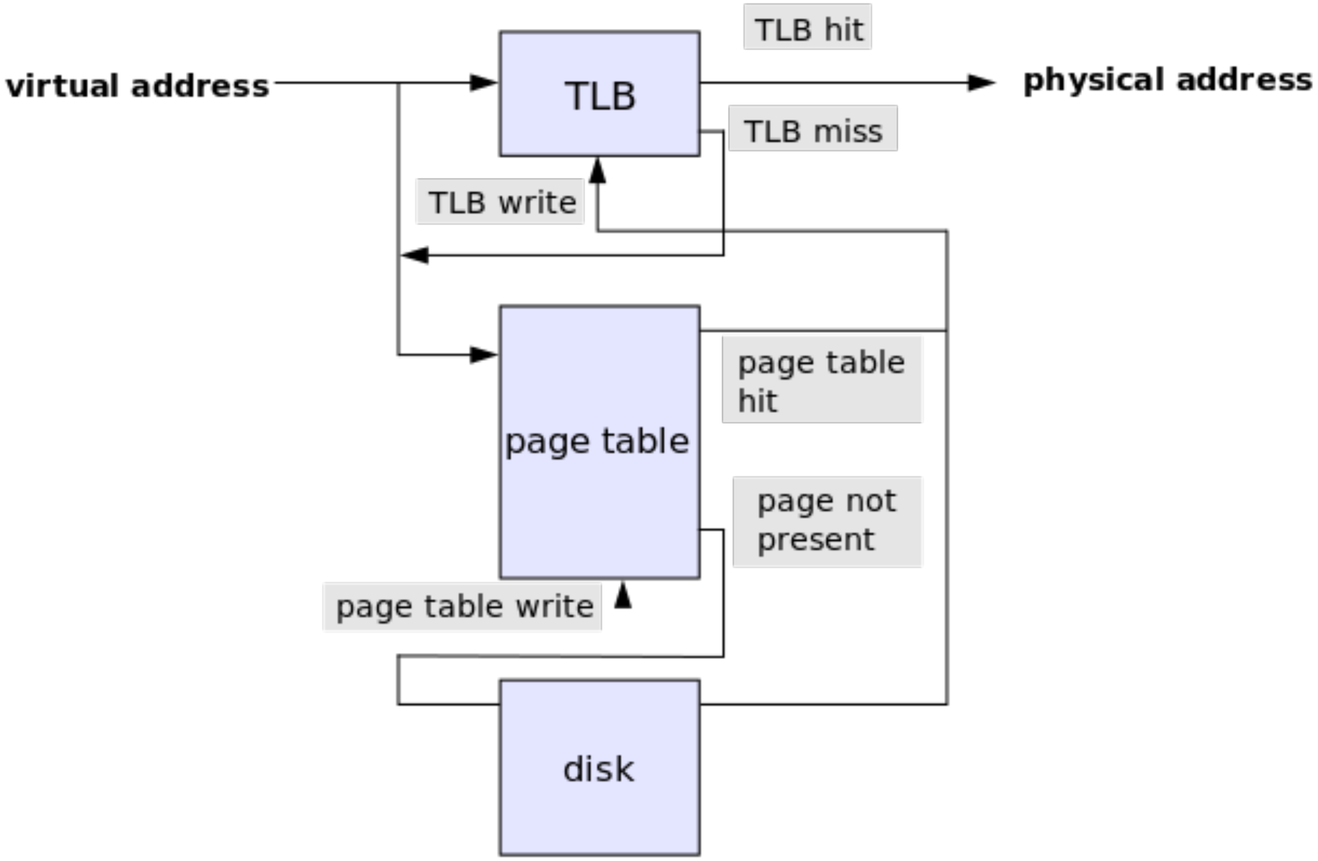
Бесплатных завтраков не бывает

Latency numbers every programmer should know

```
L1 cache reference ..........................  0.5 ns
Branch mispredict ............................    5 ns
L2 cache reference ...........................    7 ns
Mutex lock/unlock ............................   25 ns
Main memory reference ......................  100 ns
Compress 1K bytes with Zippy .............  3,000 ns  =   3 µs
Send 2K bytes over 1 Gbps network ........  20,000 ns  =  20 µs
SSD random read ........................ 150,000 ns  = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns  = 250 µs
Round trip within same datacenter ......  500,000 ns  = 0.5 ms
Read 1 MB sequentially from SSD* .....  1,000,000 ns  =   1 ms
Disk seek ............................ 10,000,000 ns  =  10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns  =  20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns  = 150 ms
```

Translation Lookaside Buffer - TLB

- Используется для ускорения трансляции виртуальных адресов в физические
- 4096 значений
- Время доступа ~ 1 цикл
- Доля пропусков 0.01 - 1 % (20-40% для разреженных данных)



Пример. Доступ к памяти

perf stat -e cycles,instructions,mem-loads,mem-stores,cache-references,cache-misses,dTLB-load-misses,dTLB-loads,dTLB-store-misses,dTLB-stores,iTLB-loads,iTLB-load-misses,page-faults,context-switches <program>

```
Performance counter stats for './latency -m 32000':

                 0      cycles                                                  (45.54%)
       383,834,148      instructions                                            (54.57%)
  <not supported>       mem-loads
        16,048,371      mem-stores                                              (54.41%)
        81,042,477      cache-references                                        (54.64%)
        61,143,093      cache-misses              #   75.446 % of all cache refs      (54.57%)
        43,167,653      dTLB-load-misses          #   35.29% of all dTLB cache hits   (54.60%)
       122,339,117      dTLB-loads                                              (36.46%)
            80,081      dTLB-store-misses                                       (36.41%)
        16,739,283      dTLB-stores                                             (36.31%)
            90,387      iTLB-loads                                              (36.43%)
           143,872      iTLB-load-misses          #  159.17% of all iTLB cache hits   (36.37%)
             8,564      page-faults
               314      context-switches

       7.413464979 seconds time elapsed
```

Пример. Распределение по памяти

Управление памятью

man brk

BRK(2) BSD System Calls Manual BRK(2)

NAME

brk, sbrk -- change data segment size

SYNOPSIS

```
#include <unistd.h>

void *
brk(const void *addr);

void *
sbrk(int incr);
```

DESCRIPTION

The brk and sbrk functions are historical curiosities left over from earlier days before the advent of virtual memory management. The brk() function sets the break or lowest address of a process's data segment (uninitialized data) to addr (immediately above bss). Data addressing is restricted between addr and the lowest stack pointer to the stack segment. Memory is allocated by brk in page size pieces; if addr is not evenly divisible by the system page size, it is increased to the next page boundary.

The current value of the program break is reliably returned by ``sbrk(0)'' (see also end(3)). The getrlimit(2) system call may be used to determine the maximum permissible size of the data segment; it will not be possible to set the break beyond the rlim_max value returned from a call to getrlimit, e.g. ``qetext + rlp->rlim_max.'' (see end(3) for the definition of etext).

RETURN VALUES

Brk returns a pointer to the new end of memory if successful; otherwise -1 with errno set to indicate why the allocation failed. The sbrk function returns a pointer to the base of the new storage if successful; otherwise -1 with errno set to indicate why the allocation failed.

Пример. brk

```c
/* Use brk syscall to allocate memory */

#include <unistd.h>
#include <malloc.h>
#include <stdlib.h>

int main() {
    char *p = sbrk(0);
    printf("p = %p\n", p);
    if(brk(p + 0x1000)) {
        perror("brk");
        return EXIT_FAILURE;
    }
    p = sbrk(0);
    printf("p = %p\n", p);
    if(brk(p + 100)) {
        perror("brk");
        return EXIT_FAILURE;
    }
    printf("*(p + 100): %x\n", *(p + 100));
    printf("*(p + 200): %x\n", *(p + 200));
    printf("*(p + 0x1000): %x\n", *(p + 0x1000)); /* segmentation fault */
    return 0;
}
```

Управление памятью

man mmap

MMAP(2) BSD System Calls Manual MMAP(2)

NAME
mmap -- allocate memory, or map files or devices into memory

LIBRARY
Standard C Library (libc, -lc)

SYNOPSIS
#include <sys/mman.h>

void *
mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset);

DESCRIPTION
The mmap() system call causes the pages starting at addr and continuing for at most len bytes to be mapped from the object described by fd, starting at byte offset offset. If offset or len is not a multiple of the pagesize, the mapped region may extend past the specified range. Any extension beyond the end of the mapped object will be zero-filled.

Пример. mmap

```c
/* Allocate memory with mmap */

#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>

int main() {
    char *p = mmap(NULL, 2000, PROT_READ | PROT_WRITE,
                   MAP_ANONYMOUS | MAP_PRIVATE, 0, 0);
    if (p == MAP_FAILED) {
        perror("mmap");
        return EXIT_FAILURE;
    }
    *(p + 2000) = 0;
    *(p + 3000) = 0;
    *(p + 4096) = 0; /* segmentation fault */
    if (-1 == munmap(p, 2000)) {
        perror("munmap");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

Источники

1. Memory Layout of Kernel and UserSpace in Linux - blogspot.com
2. Intel IA-32 manual - intel.com
3. А. В. Столяров. Программирование. Введение в профессию. Т.3 Системы и сети - stolyarov.info
4. 5-Level Paging and 5 level EPT - intel.com
5. Latency numbers every programmer should know - GitHub.com
6. Advanced Computer Concepts for the (Not So) Common Chef: Memory Hierarchy: Of Registers, Cache & Memory - intel.com
7. Playing with the perf tool in Linux - itty.ku.edu