

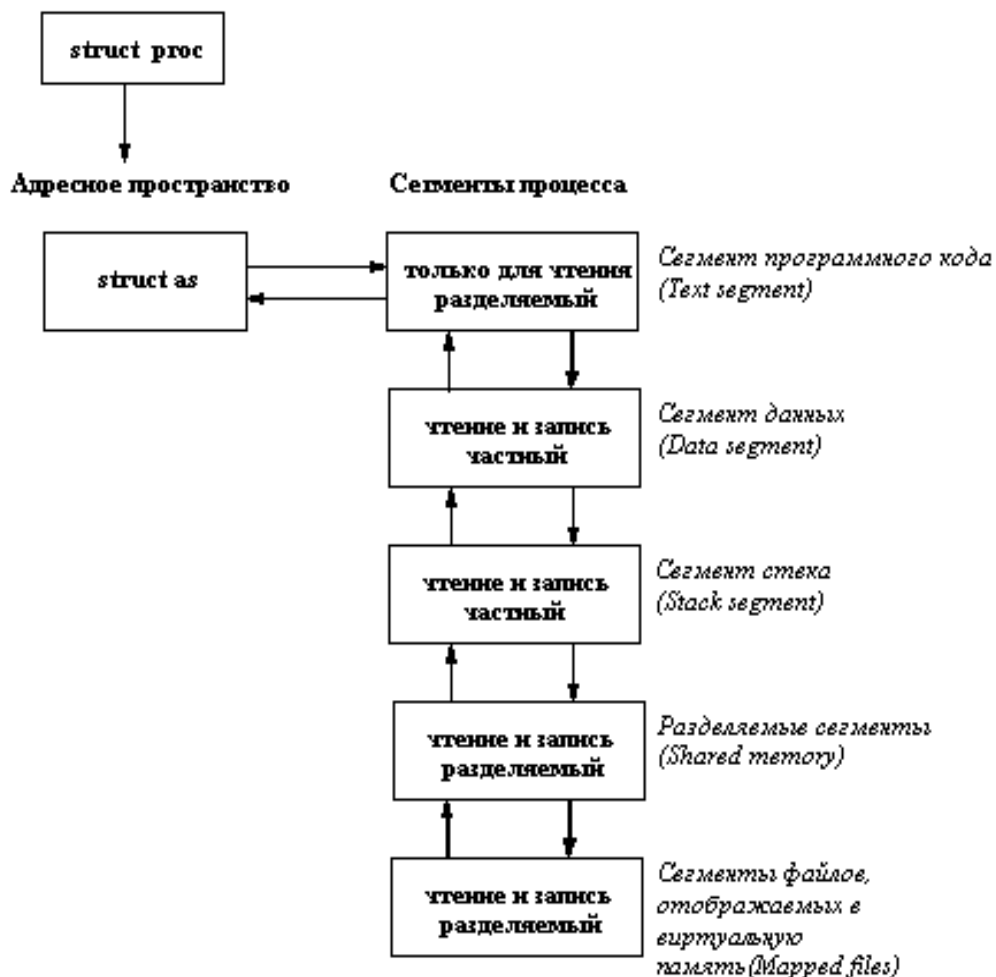
Управление памятью в ОС Linux

Основные задачи управления памятью

- Управление виртуальной памятью.
- Управление физической памятью.
- Свопинг (сброс страниц на диск) и кэширование (выделение памяти при записи данных на диск).

Сегментная структура виртуального адресного пространства

Таблица процессов



Сегмент программного кода содержит только команды, не может модифицироваться в ходе выполнения процесса

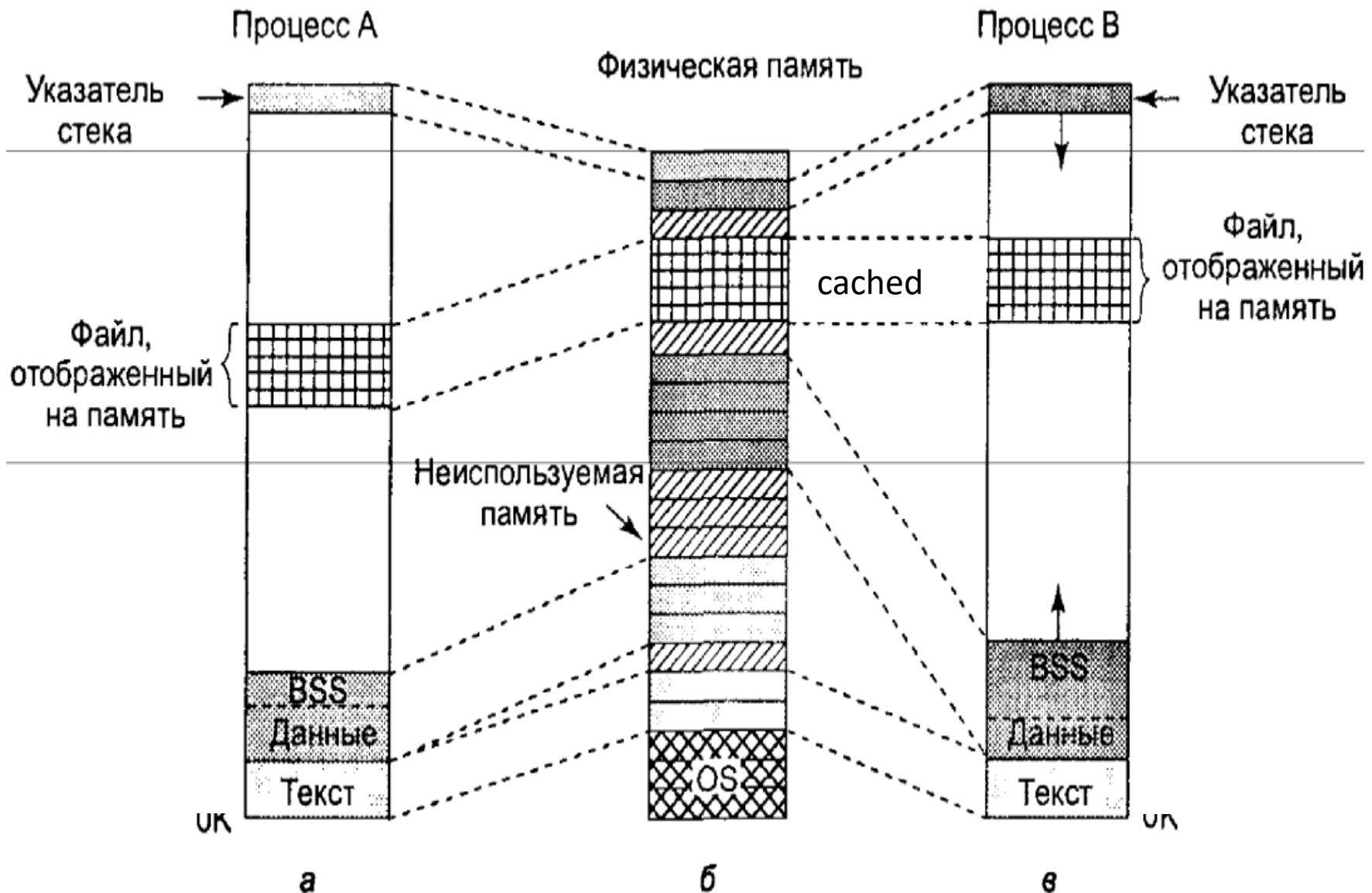
Сегмент данных содержит инициализированные и неинициализированные статические

В сегменте стека размещаются автоматические переменные программы

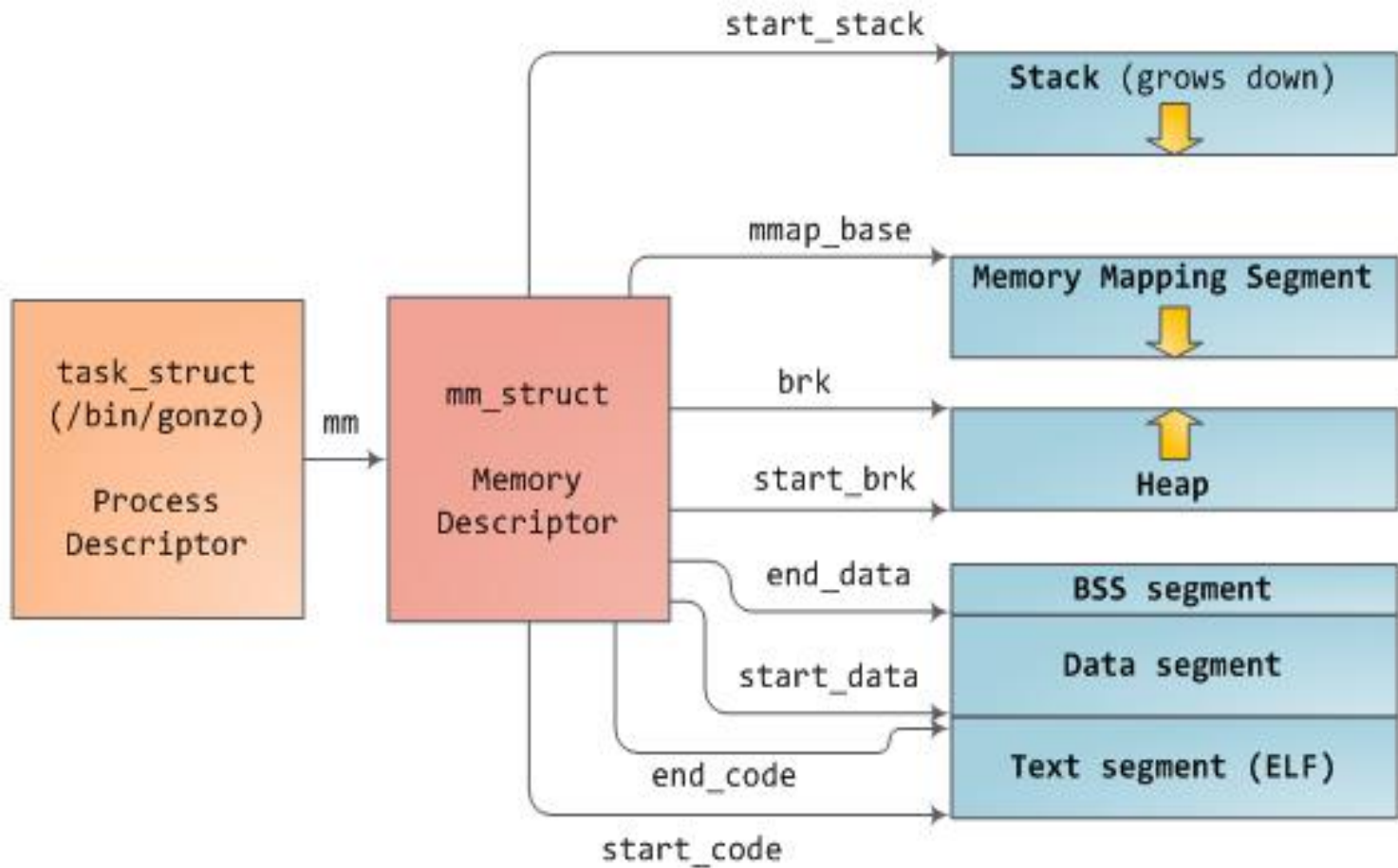
Разделяемый сегмент образуется при подключении к ней сегмента разделяемой памяти

В случае необходимости откачиваются прямо на свое место в области внешней памяти, занимаемой файлом.

Распределение виртуальной памяти пользовательского процесса



Дескриптор памяти процесса



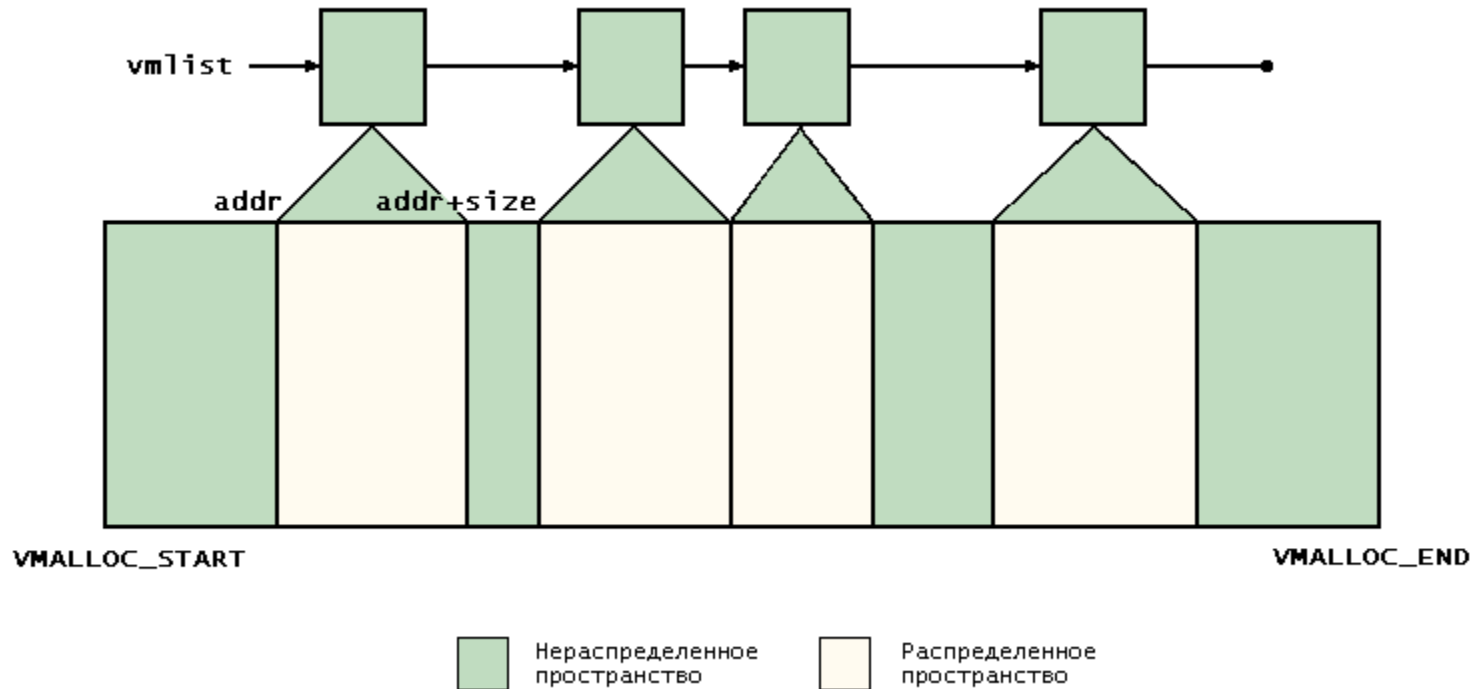
Основные поля дескриптора памяти процесса

```
struct mm_struct {
    struct vm_area_struct *mmap; /* список областей памяти */
    struct rb_root mm_rb; /* красно-черное дерево областей памяти */
    struct vm_area_struct *mmap_cache; /* последняя использованная область памяти */
    unsigned long free_area_cache; /* первый незанятый участок адресного пространства */
    pgd_t *pgd; /* глобальный каталог страниц */
    atomic_t mm_users; /* счетчик пользователей адресного пространства */
    atomic_t mm_count; /* основной счетчик использования */
    int map_count; /* количество областей памяти */
    struct rw_semaphore mmap_sem; /* семафор для областей памяти */
    spinlock_t page_table_lock; /* спин-блокировка таблиц страниц */
    struct list_head mmlist; /* список всех структур mm_struct */
    unsigned long start_code; /* начальный адрес сегмента кода */
    unsigned long end_code; /* конечный адрес сегмента кода */
    unsigned long start_data; /* начальный адрес сегмента данных */
    unsigned long end_data; /* конечный адрес сегмента данных */
    unsigned long start_brk; /* начальный адрес сегмента "кучи" */
    unsigned long brk; /* конечный адрес сегмента "кучи" */
    unsigned long start_stack; /* начало стека процесса */
    unsigned long arg_start; /* начальный адрес области аргументов */
    unsigned long arg_end; /* конечный адрес области аргументов */
    unsigned long env_start; /* начальный адрес области переменных среды */
    unsigned long env_end; /* конечный адрес области переменных среды */
    unsigned long rss; /* количество физических страниц памяти */
    unsigned long total_vm; /* общее количество страниц памяти */
    unsigned long locked_vm; /* количество заблокированных страниц памяти */
    unsigned long def_flags; /* флаги доступа, используемые по умолчанию */
    unsigned long cpu_vm_mask; /* маска отложенного переключения буфера TLB */
    unsigned long swap_address; /* последний сканированный адрес */
    unsigned dumpable:1; /* можно ли создавать файл core? */
    int used_hugetlb; /* используются ли гигантские страницы памяти (hugetlb)? */
    int core_waiters; /* количество потоков, ожидающих на создание файла core */
    struct completion *core_startup_done; /* условная переменная начала создания файла core */
    struct completion core_done; /* условная переменная завершения создания файла core */
};
```

Управление виртуальной памятью

1. Ядро создает новое виртуальное адресное пространство при создании нового процесса системным вызовом **fork** и когда процесс запускает новую программу системным вызовом **exec**.
2. Создание нового процесса с помощью **fork** включает создание полной копии адресного пространства существующего процесса.
3. Каждый процесс считает, что ей выделен непрерывный участок виртуальной памяти максимального размера, поддерживаемого на соответствующей архитектуре.
4. Ядро выделяет виртуальную память страницами фиксированного размера.
5. Выделение физической страницы осуществляется в момент обращения к виртуальной странице.
6. Если свободных страниц больше нет, но существует файл подкачки, куда ядро может убрать одну из наиболее долго не использовавшихся страниц, и освободившуюся физическую страницу отдать запросившему память процессу.

Учет виртуальной памяти



```
struct vm_struct {  
    unsigned long flags;  
    void * addr;  
    unsigned long size;  
    struct vm_struct * next;  
};
```

Выделить виртуальные страницы памяти

`void * vmalloc(unsigned long size)`

Освободить виртуальные страницы памяти

`void vfree(void * addr)`

В 32-битной архитектуре максимальная адресуемая память составляет 4GB,
в 64-битной архитектуре адресного пространства - 4TB

Мониторинг виртуальной памяти

Команда free	Единица измерения - 1 страница памяти (4 Кб)					
	total	used	free	shared	buffers	cached
Память:	1024988	864824	160164	5872	69708	434108
-/+ буферы/кэш:		361008	663980			

(показывается, сколько памяти используется и сколько памяти свободно с точки зрения ее использования в приложениях.)

Swap (подкачка): 522236 0 522236

Параметры использования памяти можно увидеть в файлах:

/proc/meminfo - подробная информация о памяти,

/proc/sys/vm/swappiness уровень свободной памяти, при котором система начнет активно сбрасывать память в своп (60%).

/proc/sys/vm/vfs_cache_pressure - уровень выделяемой памяти под кэш (100), чем больше, тем активнее выгружаются неиспользуемые страницы памяти из кеша.

/proc/sys/vm/overcommit_memory — стратегия перевыделение памяти (0)

/proc/sys/vm/overcommit_ratio уровень разрешения на перевыделение памяти (50%) для стратегии 2

Overcommit — стратегия выделения памяти, когда операционная система разрешает приложениям занимать больше виртуальной памяти, чем доступно в системе. Используется в Linux по умолчанию.

OVERCOMMIT_GUESS 0 — эвристический подход к распределению памяти. Система будет отвергать только запросы, которые в принципе не могут быть удовлетворены, остальные — удовлетворять вне зависимости от наличия свободной памяти.

OVERCOMMIT_ALWAYS 1 — ядро всегда удовлетворяет любые запросы на выделение памяти.

OVERCOMMIT_NEVER 2 — система всегда будет выделять память только если она подкреплена реальными страницами в ОЗУ или свопе.

Ограничение системных ресурсов

Утилита **ulimit** позволяет просмотреть и ограничить системные ресурсы.

Каждый пользователь может уменьшить собственный лимит, но только суперпользователь может его увеличить.

gena@gena-VirtualBox:~\$ ulimit -a отображает текущие установки

core file size (blocks, -c) 0 , размер core файла

data seg size (kbytes, -d) unlimited, максимальный размер сегмента данных

scheduling priority (-e) 0 статический приоритет процесса

file size (blocks, -f) unlimited , максимальный размер создаваемого файла

pending signals (-i) 7858 максимальное число отложенных сигналов

max locked memory (kbytes, -l) 64 максимальный размер резидентной части процесса, находящейся в ОЗУ

max memory size (kbytes, -m) unlimited

open files (-n) 1024, максимальное число открытых файлов

pipe size (512 bytes, -p) 8 , размер буфера канала

POSIX message queues (bytes, -q) 819200 максимальный размер очереди сообщений

real-time priority (-r) 0 приоритет реального времени (не задан)

stack size (kbytes, -s) 8192 максимальный размер стека

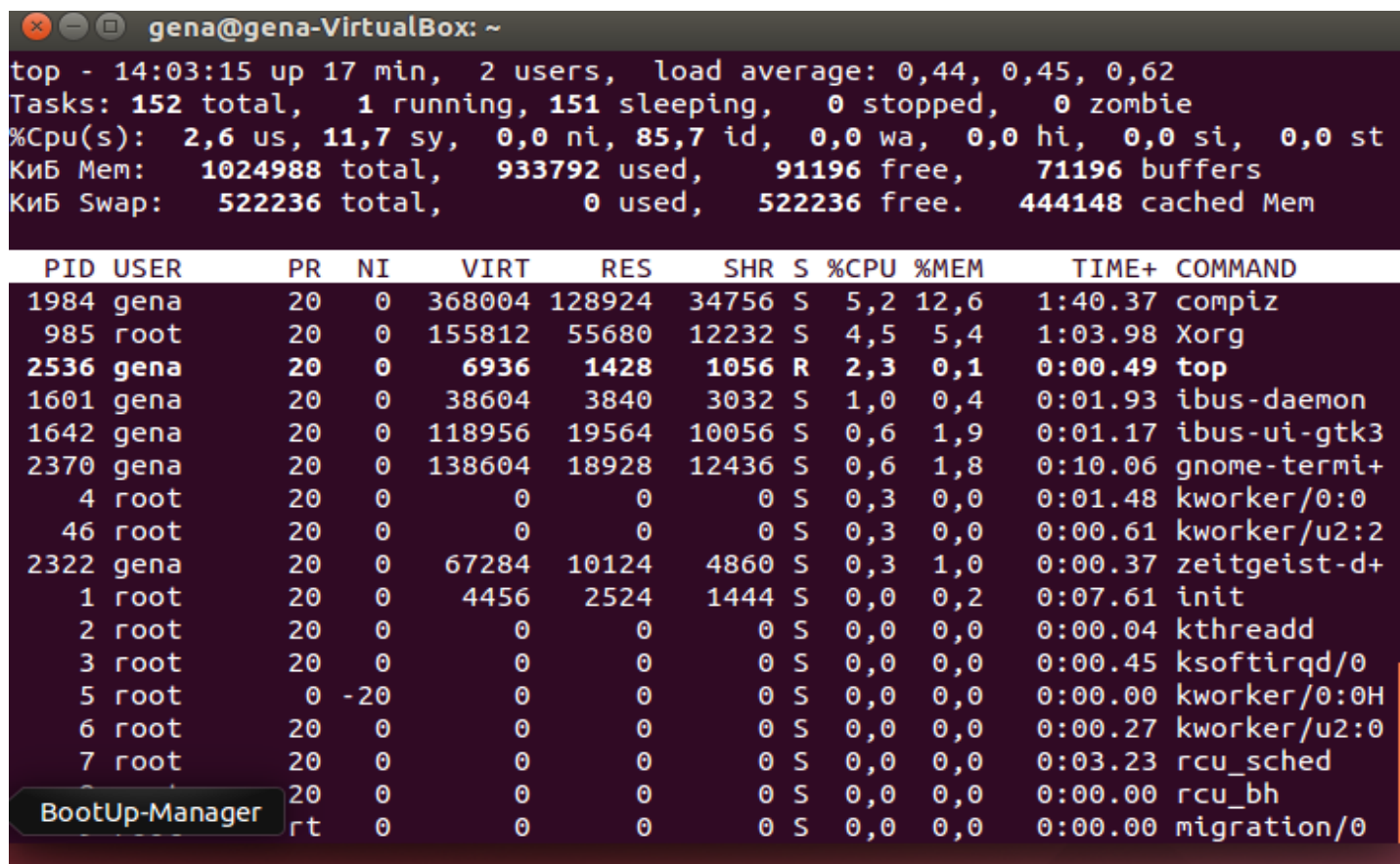
cpu time (seconds, -t) unlimited максимальное время работы процесса

max user processes (-u) 7858, максимальное число запущенных пользователем процессов

virtual memory (kbytes, -v) unlimited, максимальный размер используемой виртуальной памяти

file locks (-x) unlimited

Информация команды top по процессам



```
gena@gena-VirtualBox: ~
top - 14:03:15 up 17 min,  2 users,  load average: 0,44, 0,45, 0,62
Tasks: 152 total,   1 running, 151 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2,6 us, 11,7 sy,   0,0 ni, 85,7 id,   0,0 wa,   0,0 hi,   0,0 si,   0,0 st
КиБ Mem:  1024988 total,  933792 used,   91196 free,   71196 buffers
КиБ Swap:  522236 total,    0 used,   522236 free.  444148 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1984	gena	20	0	368004	128924	34756	S	5,2	12,6	1:40.37	compiz
985	root	20	0	155812	55680	12232	S	4,5	5,4	1:03.98	Xorg
2536	gena	20	0	6936	1428	1056	R	2,3	0,1	0:00.49	top
1601	gena	20	0	38604	3840	3032	S	1,0	0,4	0:01.93	ibus-daemon
1642	gena	20	0	118956	19564	10056	S	0,6	1,9	0:01.17	ibus-ui-gtk3
2370	gena	20	0	138604	18928	12436	S	0,6	1,8	0:10.06	gnome-termi+
4	root	20	0	0	0	0	S	0,3	0,0	0:01.48	kworker/0:0
46	root	20	0	0	0	0	S	0,3	0,0	0:00.61	kworker/u2:2
2322	gena	20	0	67284	10124	4860	S	0,3	1,0	0:00.37	zeitgeist-d+
1	root	20	0	4456	2524	1444	S	0,0	0,2	0:07.61	init
2	root	20	0	0	0	0	S	0,0	0,0	0:00.04	kthreadd
3	root	20	0	0	0	0	S	0,0	0,0	0:00.45	ksoftirqd/0
5	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	kworker/0:0H
6	root	20	0	0	0	0	S	0,0	0,0	0:00.27	kworker/u2:0
7	root	20	0	0	0	0	S	0,0	0,0	0:03.23	rcu_sched
		20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_bh
	BootUp-Manager	rt	0	0	0	0	S	0,0	0,0	0:00.00	migration/0

VIRT указывает, сколько виртуальной памяти в настоящий момент доступно процессу.

RES указывает, сколько в действительности потребляется процессом реальной физической памяти.

SHR показывает, какая величина от значения VIRT является в действительности разделяемой памятью.

%MEM - процент использования общей оперативной памяти

Детальная информация о процессе

cat /proc/PID/status

Name: bash
State: S (sleeping)
Tgid: 2595
Ngid: 0
Pid: 2595
PPid: 2585
TracerPid: 0
Uid: 1000 1000 1000 1000
Gid: 1000 1000 1000 1000
FDSize: 256
Groups: 4 24 27 30 46 112 124 1000
VmPeak: 8380 kB пиковый размер виртуальной памяти
VmSize: 8380 kB текущий размер виртуальной памяти
VmLck: 0 kB объем заблокированной памяти (размещается в ОЗУ)
VmPin: 0 kB размер закрепленной памяти (страницы не могут быть перемещены)
VmHWM: 3228 kB пиковый размер физической памяти
VmRSS: 3228 kB текущий размер физической памяти
VmData: 1440 kB размер сегмента данных
VmStk: 136 kB размер стека
VmExe: 944 kB размер сегмента кода
VmLib: 2156 kB размер общих библиотек
VmPTE: 32 kB размер таблицы страниц
VmSwap: 0 kB используемый объем для подкачки

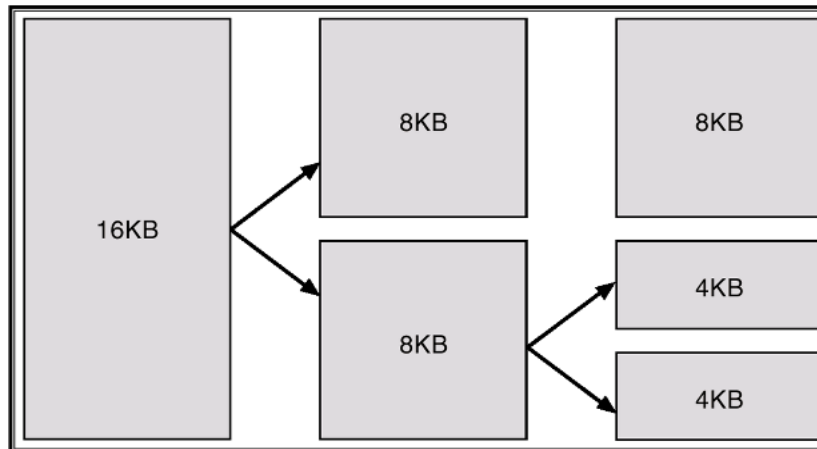
.....

Управление физической памятью

Система распределения физической памяти в Linux занимается размещением и освобождением страниц, группы страниц и небольших блоков памяти.

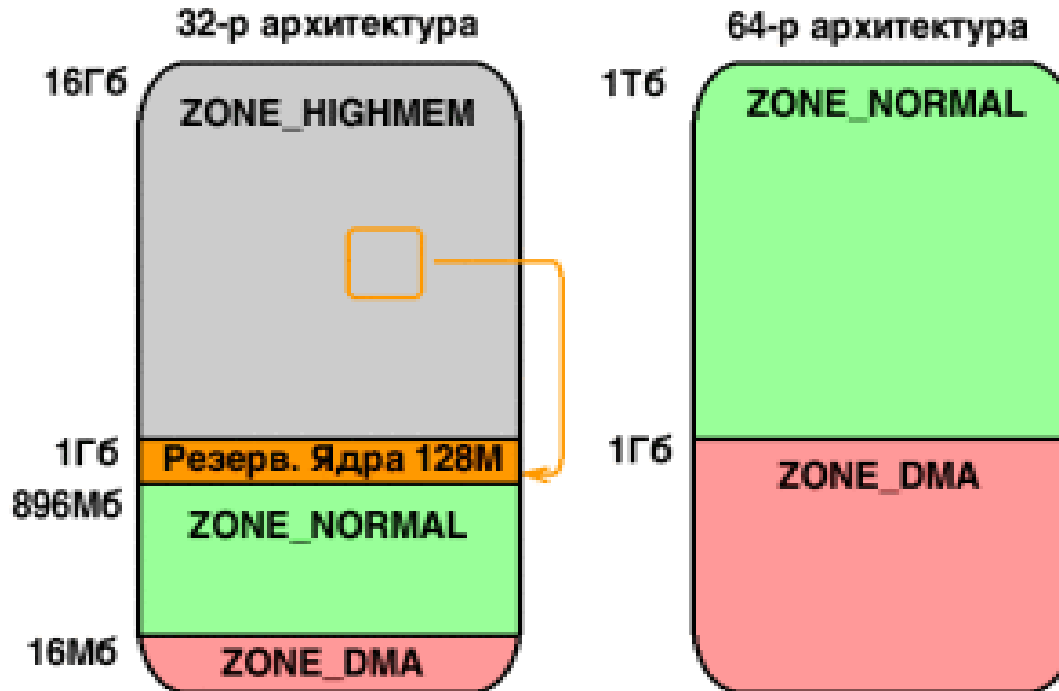
Распределитель страниц Linux использует алгоритм **buddy-heap (партнерской кучи)** для слежения за доступными физическими страницами, принципы которого в следующем:

- Каждая область памяти, подлежащая распределению, образует пару с ее смежным "партнером".
- Когда обе области-партнера освобождаются, они сливаются и образуют смежную область вдвое большего размера.
- Если не существует малой области памяти, чтобы удовлетворить небольшой запрос на память, то область памяти большего размера расщепляется на две области-партнера для удовлетворения данного запроса.



Зоны памяти

Ядро делит всю доступную физическую память на 3 зоны:



Архитектура памяти Linux
Максимальный объем указан для ARMv8 (https://www.redhat.com/en/tech-tips-compared)

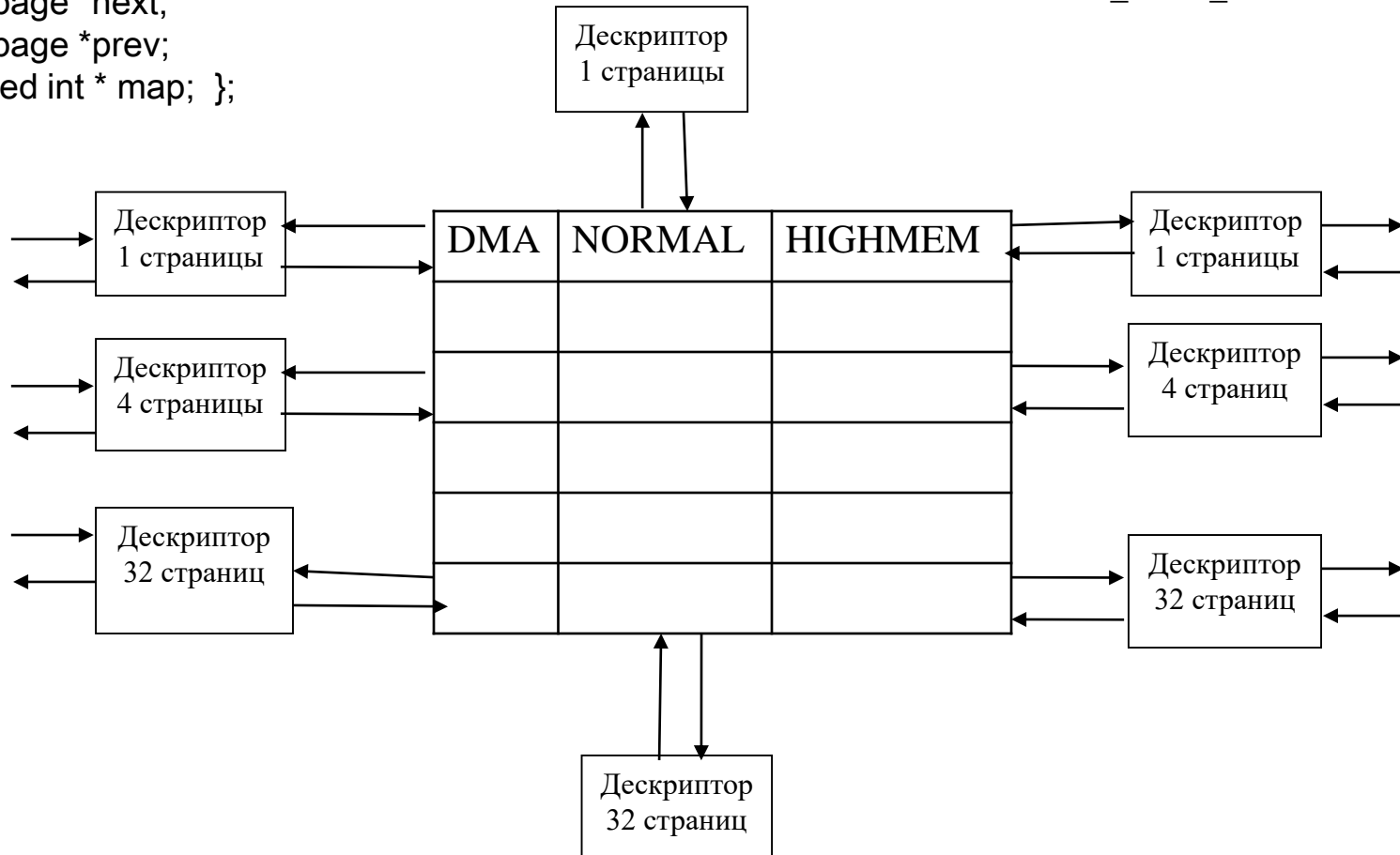
- **ZONE_DMA.** Страницы, совместимые с режимом DMA (обмена данными между устройствами компьютера или же между устройством и основной памятью, в котором центральный процессор не участвует).
- **ZONE_NORMAL.** Страницы, которые отображаются в адресные пространства пользователя обычным способом.
- **ZONE_HIGHMEM.** "Верхняя память", содержащая страницы, которые не могут отображаться в адресное пространство ядра.

Список свободных страниц памяти

```
static struct free_area_struct free_area[NR_MEM_TYPES][NR_MEM_LISTS];
```

```
struct free_area_struct {  
    struct page *next;  
    struct page *prev;  
    unsigned int *map; };
```

NR_MEM_LISTS 6 или 12



Дескриптор физической страницы

```
typedef struct page {  
    struct page *next;  
    struct page *prev;  
    struct inode *inode;  
    unsigned long offset;  
    struct page *next_hash;  
    atomic_t count; /* счетчик использования страницы, кол-во процессов,  
                     имеющих доступ к странице */  
    unsigned flags; /* состояние страницы: страница  
                     заблокирована, произошла ошибка, страница активна, нельзя  
                     выгружать ..... */  
    struct wait_queue *wait;  
    struct page *prev_hash;  
    struct buffer_head * buffers;  
    unsigned long swap_unlock_entry;  
    unsigned long map_nr; /* page->map_nr == page - mem_map */  
} mem_map_t;
```


Выделение и освобождение страниц памяти

- Выделение одной страницы с заполненными нулями
`get_zeroed_page(unsigned int flags);`

- Выделение одной страницы без очистки
`__get_free_page(unsigned int flags);`

- Выделение несколько страниц без очистки
`__get_free_pages(unsigned int flags, unsigned int order);`

Аргумент flags: `__GFP_DMA`, `__GFP_HIGHMEM`

order (порядок) $\log_2 N$ N- число страниц

- Освобождение одной страницы
`void free_page(unsigned long addr);`

- Освобождение несколько страниц
`void free_pages(unsigned long addr, unsigned long order);`

Блочная организация памяти

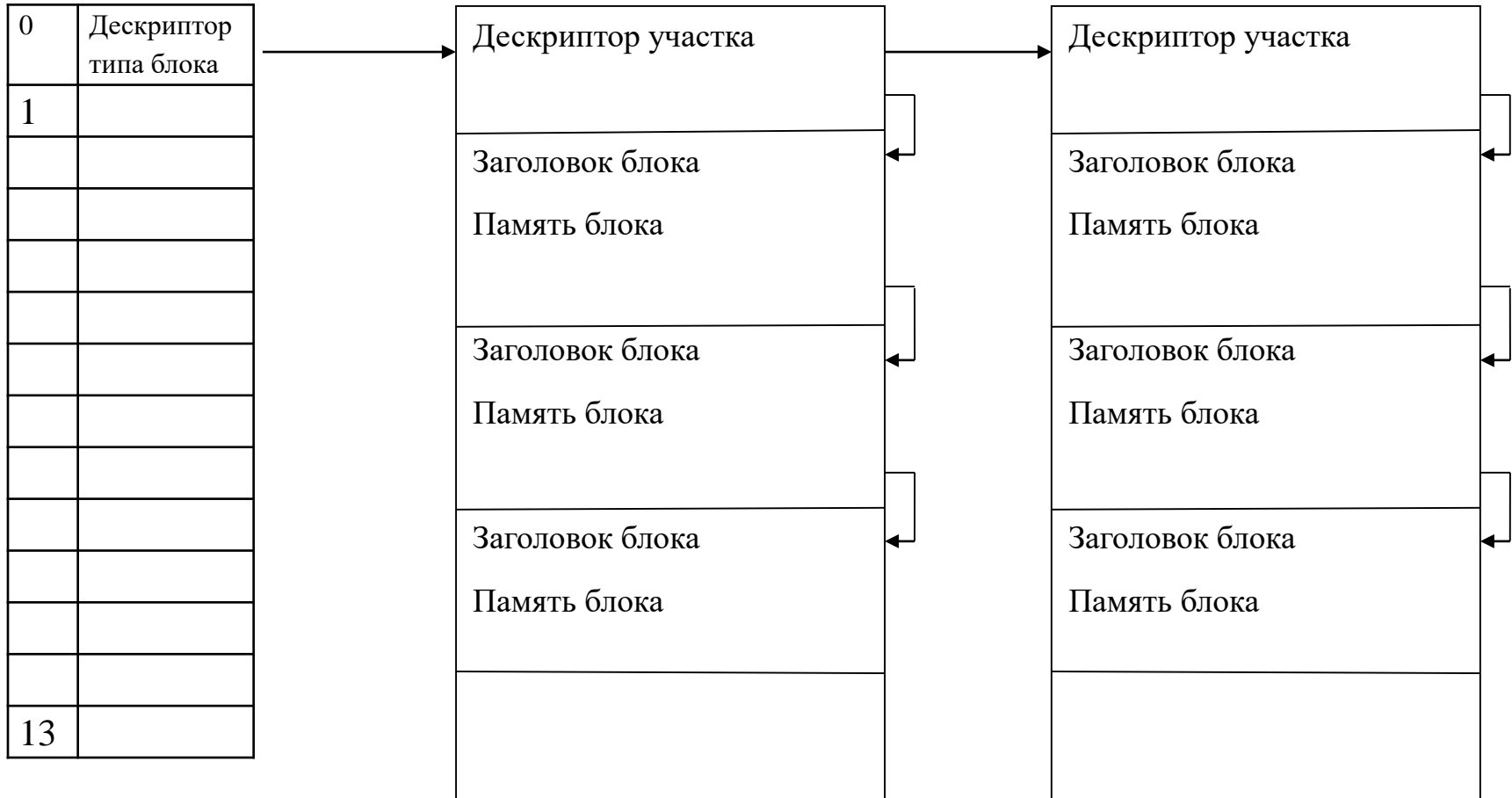
- Доступное ядру пространство ограничено 1Гб виртуальной и физической памяти.
- Размеры структур данных ядра меньше страницы
- Память ядра не выгружается.
- Часто ядро требует физически непрерывных регионов памяти.
- Зачастую ядро должно выделять память, не засыпая.

Размеры блоков зависят от архитектуры процессора
static const unsigned int
blocksize[] = {

32, (24)
64, (56)
128, (120)
252, (244)
508, (500)
1020, (1012)
2040, (2032)
4080, (4072)
8176, (8168)
16368, (16360)
32752, (32744)
65520, (65512)
131056, (131048)
0

};

Схема блочной организации памяти



Дескриптор блока

Дескриптор блока

```
struct size_descriptor {  
    struct page_descriptor *firstfree; /*указатель на  
    первый свободный блок */  
    struct page_descriptor *dmafree; /*  
    указатель на первый свободный блок DMA-  
    able memory */  
  
    int nblocks; /*число блоков на участке*/  
    int nmallocs; /*число занятых блоков */  
    int nfrees; /*число свободных блоков */  
    int nbytesmalloced; /*объем занятой памяти */  
    int npages; /*число выделенных страниц */  
    unsigned long gfporder; /* номер типа участка  
    */
```

```
static struct size_descriptor sizes[] =  
{  
    {NULL, NULL, 127, 0, 0, 0, 0, 0},  
    {NULL, NULL, 63, 0, 0, 0, 0, 0},  
    {NULL, NULL, 31, 0, 0, 0, 0, 0},  
    {NULL, NULL, 16, 0, 0, 0, 0, 0},  
    {NULL, NULL, 8, 0, 0, 0, 0, 0},  
    {NULL, NULL, 4, 0, 0, 0, 0, 0},  
    {NULL, NULL, 2, 0, 0, 0, 0, 0},  
    {NULL, NULL, 1, 0, 0, 0, 0, 0},  
    {NULL, NULL, 1, 0, 0, 0, 0, 1},  
    {NULL, NULL, 1, 0, 0, 0, 0, 2},  
    {NULL, NULL, 1, 0, 0, 0, 0, 3},  
    {NULL, NULL, 1, 0, 0, 0, 0, 4},  
    {NULL, NULL, 1, 0, 0, 0, 0, 5},  
    {NULL, NULL, 0, 0, 0, 0, 0, 0}  
};
```

Структуры данных для блочной организации памяти

Дескриптор участка

```
struct page_descriptor { /* указатель на следующую область */
    struct page_descriptor *next;
/* указатель на первый свободный блок */
    struct block_header *firstfree;
    int order; /* номер типа участка */
    int nfree; /* число свободных блоков */
};
```

Заголовок блока

```
struct block_header {
    unsigned long bh_flags; /* флаги блока */
    union {
        /* число занятых байтов */
        unsigned long ubh_length;
/* указатель на следующий свободный блок */
        struct block_header *fbh_next;
    } vp;
};
```

Выделение и освобождение блока

Выделение памяти

```
void *kmalloc(size_t size, int flags);
```

Флаги:

- GFP_ATOMIC Используется для выделения памяти в обработчиках прерываний и другом коде вне контекста процесса.
- GFP_KERNEL Выделение производится от имени процесса, который выполняет системный запрос в пространстве ядра.
- GFP_USER Используется для выделения памяти для страниц пространства пользователя.

Освобождение памяти

```
void kfree( const void *ptr );
```