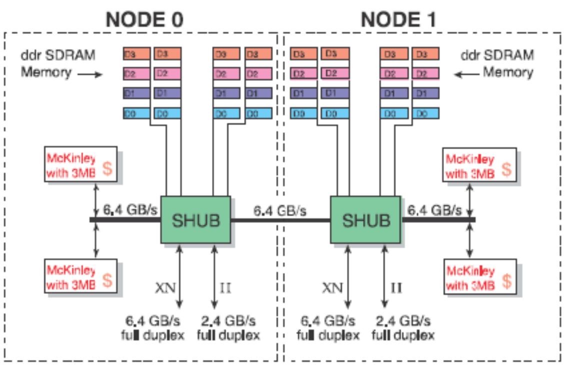
Memory Management

1. Numa

what's a numa?

Non-Uniform Memory Access (NUMA) refers to multiprocessor systems whose memory is divided into multiple memory nodes. The access time of a memory node depends on the relative locations of the accessing CPU and the accessed node.



How to check numa status?

[root@pek2-office-18th-10-117-174-219 ~]# numactl --hardware
available: 1 nodes (0)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 15994 MB
node 0 free: 226 MB
node distances:
node 0
0: 10

Numa migration

Move pages from one node to another. system call is provided by kernel.

NAME
migrate_pages - move all pages in a process to another set of nodes
SYNOPSIS
#include <numaif.h>
long migrate_pages(int pid, unsigned long maxnode,
const unsigned long *old_nodes,
const unsigned long *new_nodes);

https://www.kernel.org/doc/Documentation/vm/page_migration

2.Zone

What's a zone?

Kernel divided physical memory with in a numa node into zones based on various restrictions on how it can be used.

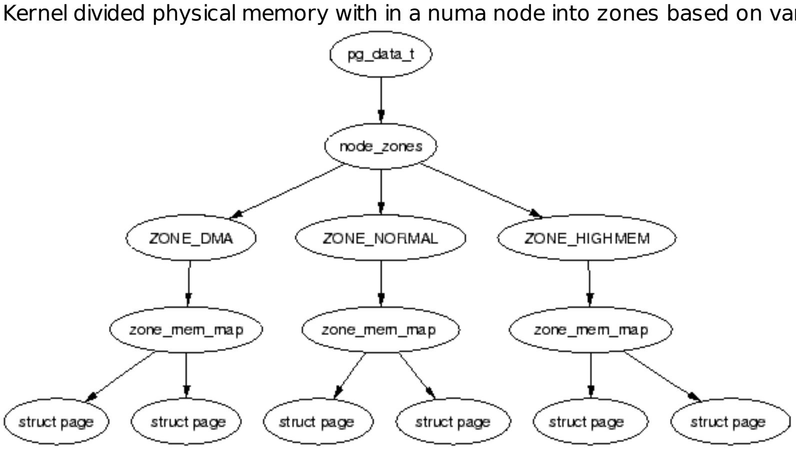


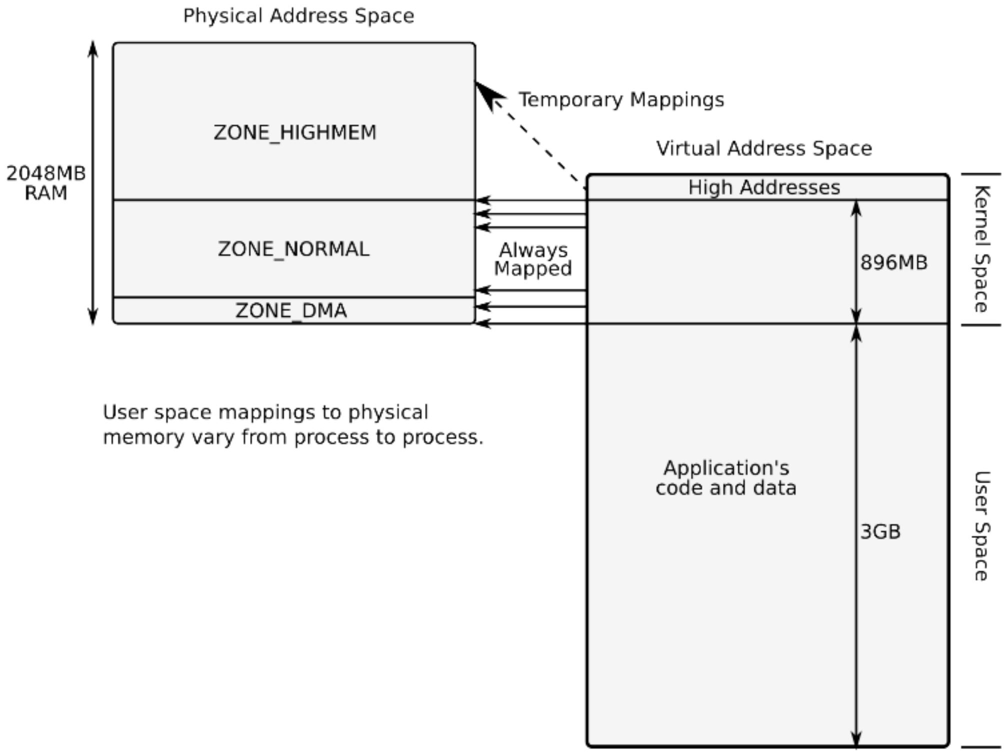
Figure 2.1: Relationship Between Nodes, Zones and Pages

How about Android kernel?

Yes, it has only normal and high zone!!!

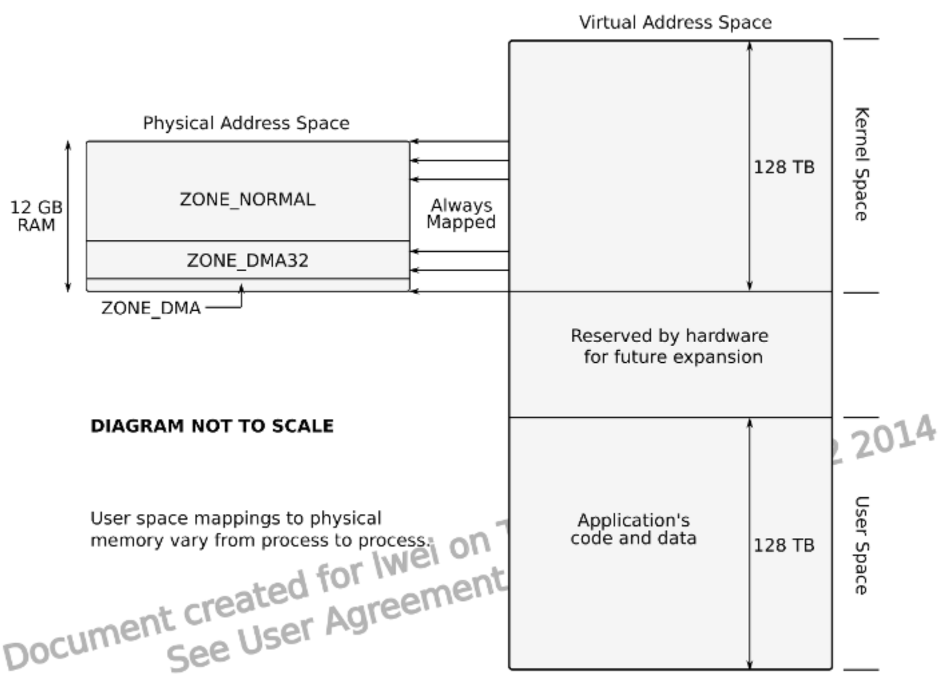
```
127|u0_a150@android:/ $ cat /proc/buddyinfo
Node 0, zone   Normal   7506   1472   118   16   12
Node 0, zone  HighMem    338     41     1    0    3
u0_a150@android:/ $
```

X86 32 address space mapping



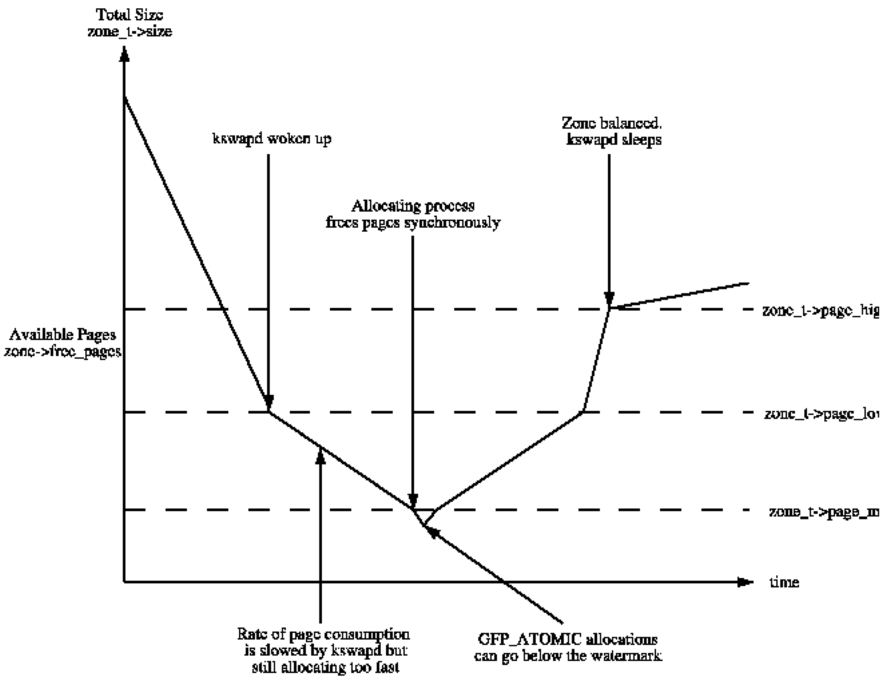
X86-64 address space mapping

Figure 10.1. x86-64 virtual address space



http://www.ibm.com/developerworks/cn/linux/l-numa/

watermark of a zone



Emergent Pool

#define ALLOC_HIGH 0x20 /* __GFP_HIGH set */
 if (alloc_flags & ALLOC_HIGH)
 min -= min / 2;

low mem reserve

low mem reserve is used protect the low memory zone by raising watermark up when there's failure in up level zone.

How to check check the workmark of zone?

```
[gliu@pek2-office-18th-10-117-174-219 ~]$ cat /proc/zoneinfo
Node 0, zone      DMA
  pages free     3871
        min      16
        low      20
        high     24
        scanned  0
        spanned  4095
        present  3998
        managed  3977
```

or

echo m >/proc/sysrq-trigger ; tail /var/log/message -n 100

```
crash> kmem -n
NODE    SIZE      PGLIST_DATA       BOOTMEM_DATA       NODE_ZONES
  0    524285   ffff88000000a000  ffffffff81cae100  ffff88000000a000
                                                    ffff8800000126c0
                                                    ffff88000001ad80
                                                    ffff880000023440
    MEM_MAP          START_PADDR    START_MAPNR
ffffea0000000038        1000             1


ZONE  NAME         SIZE       MEM_MAP      START_PADDR  START_MAPNR
```

```
  0   DMA         4095  ffffea0000000038         1000           1
  1   DMA32      520190  ffffea0000038000      1000000        4096
  2   Normal         0                0           0           0
  3   Movable        0                0           0           0
```

```
crash> kmem -z
NODE: 0  ZONE: 0  ADDR: ffff88000000a000  NAME: "DMA"
  SIZE: 4095  PRESENT: 3841  MIN/LOW/HIGH: 83/103/124
  VM_STAT:
        NR_FREE_PAGES: 3938

NODE: 0  ZONE: 1  ADDR: ffff8800000126c0  NAME: "DMA32"
  SIZE: 520190  PRESENT: 513078  MIN/LOW/HIGH: 11180/13975/16770
  VM_STAT:
        NR_FREE_PAGES: 270234
```

Tuning of min watermark and low memory reserve?
vm.min_free_kbytes
vm.lowmem_reserve_ratio

3.page

data struct of a page

```
struct page {
    long unsigned int flags;  <---page flags, like lru/dirty/writeback/active/inactive/slab/highmem...
    atomic_t _count;    <----how many processes uses this page?
                                     checked this counter when dropping a cache.
    union {
        atomic_t _mapcount;  <---how many page table(pte) points to this page
        struct {
            u16 inuse;
            u16 objects;
        };
    };
    union {
        struct {
            long unsigned int private;
            struct address_space *mapping;      <-----if this page is file backed, it points to the address_space,
                                                          if it's anon page, points anon_struct
        };
        spinlock_t ptl;
        struct kmem_cache *slab;
        struct page *first_page;
    };
    union {
        long unsigned int index;   <------if this page if file backed, index is used to store the offset in this file,
                                            mean the position of this page within a file.
        void *freelist;
    };
    struct list_head lru;   <---used for PFRA
}
```

mem_map
This is a array to store all the physical page infomation.

zone has a index to store where this zone starts at mem_map

```
crash> whatis zone.zone_start_pfn
struct zone {
  [34336] long unsigned int zone_start_pfn;
}
```

mem_map array take about 1.36% of total memory
mem_map ratio = sizeof(page)/4096

4. buddy allocator

what's buddy allocator?

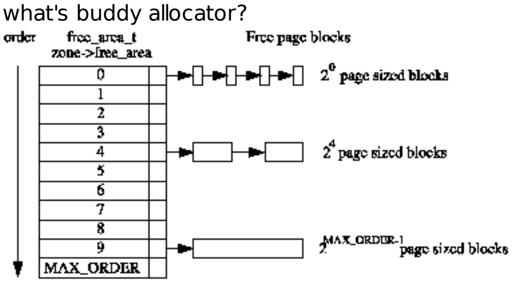


Figure 6.1: Free page block management

buddy allocator interface

`struct page * alloc_page(unsigned int gfp_mask)`
�Allocate a single page and return a struct address
`struct page * alloc_pages(unsigned int gfp_mask, unsigned int order)`
�Allocate 2^{order} number of pages and returns a struct page
`unsigned long get_free_page(unsigned int gfp_mask)`
�Allocate a single page, zero it and return a virtual address
`unsigned long __get_free_page(unsigned int gfp_mask)`
�Allocate a single page and return a virtual address
`unsigned long __get_free_pages(unsigned int gfp_mask, unsigned int order)`
�Allocate 2^{order} number of pages and return a virtual address
`struct page * __get_dma_pages(unsigned int gfp_mask, unsigned int order)`
�Allocate 2^{order} number of pages from the DMA zone and return a struct page

GFP flag

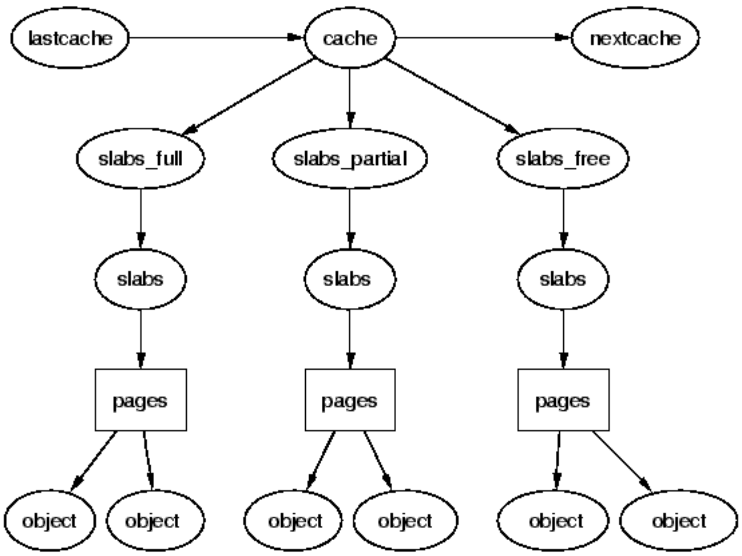
Flag	Description
`__GFP_WAIT`	Indicates that the caller is not high priority and can sleep or reschedule
`__GFP_HIGH`	Used by a high priority or kernel process. Kernel 2.2.x used it to determine if a process could access emergency pools of memory. In 2.4.x kernels, it does not appear to be used
`__GFP_IO`	Indicates that the caller can perform low level IO. In 2.4.x, the main affect this has is determining if `try_to_free_buffers()` can flush buffers or not. It is used by at least one journaled filesystem
`__GFP_HIGHIO`	Determines that IO can be performed on pages mapped in high memory. Only used in `try_to_free_buffers()`
`__GFP_FS`	Indicates if the caller can make calls to the filesystem layer. This is used when the caller is filesystem related, the buffer cache for instance, and wants to avoid recursively calling itself

Flag	Low Level Flag Combination
`GFP_ATOMIC`	HIGH
`GFP_NOIO`	HIGH | WAIT
`GFP_NOHIGHIO`	HIGH | WAIT | IO
`GFP_NOFS`	HIGH | WAIT | IO | HIGHIO

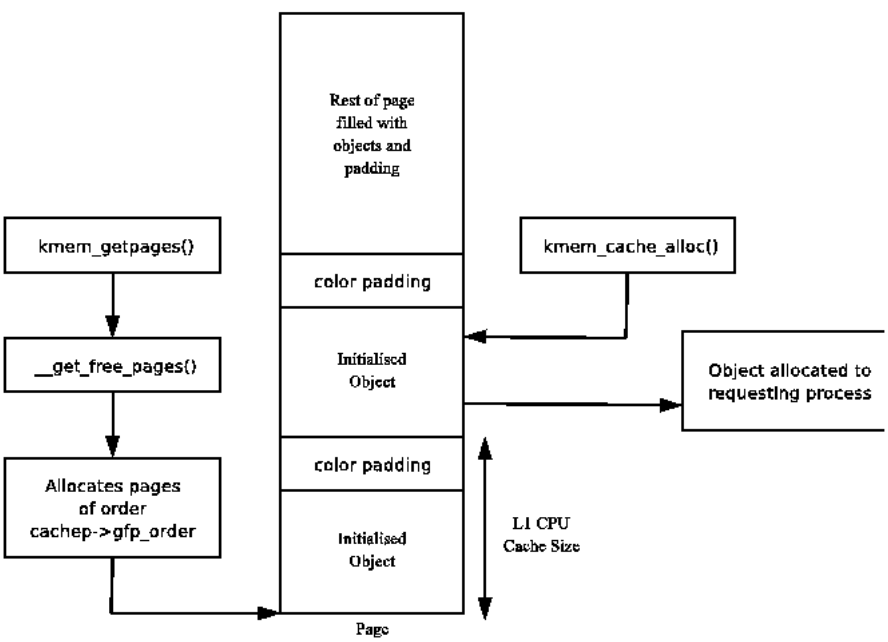
GFP_KERNEL	HIGH | WAIT | IO | HIGHIO | FS
GFP_NFS	HIGH | WAIT | IO | HIGHIO | FS
GFP_USER	WAIT | IO | HIGHIO | FS
GFP_HIGHUSER	WAIT | IO | HIGHIO | FS | HIGHMEM
GFP_KSWAPD	WAIT | IO | HIGHIO | FS

Avoid memory fragmentation???
1. memory compaction
http://lwn.net/Articles/368869/
2. zone_movable
http://lwn.net/Articles/224255/
3. lumpy reclaim
http://lwn.net/Articles/211199/

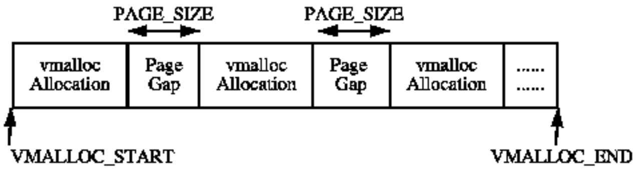
5. slab allocator

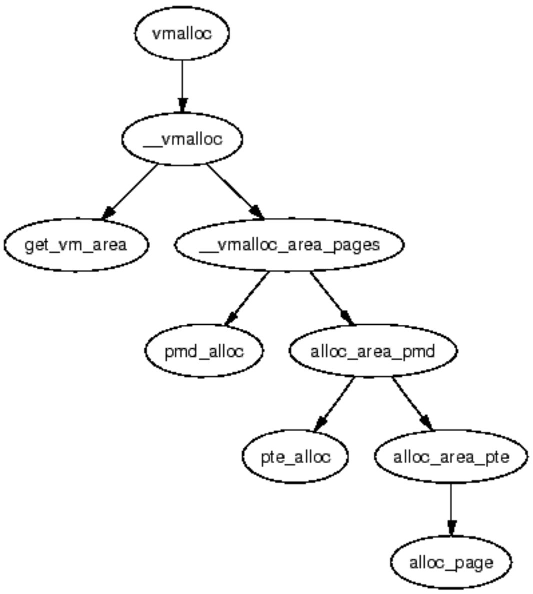


Color of slab?
In this scenario, the first slab created will have its objects start at 0. The second will start at 32, the third at 64, the fourth at 96 and the fifth will start back at 0. With this, objects from each of the slabs will not hit the same hardware cache line on the CPU. The value of `colour` is 3 and `colour_off` is 32.



6. vmalloc

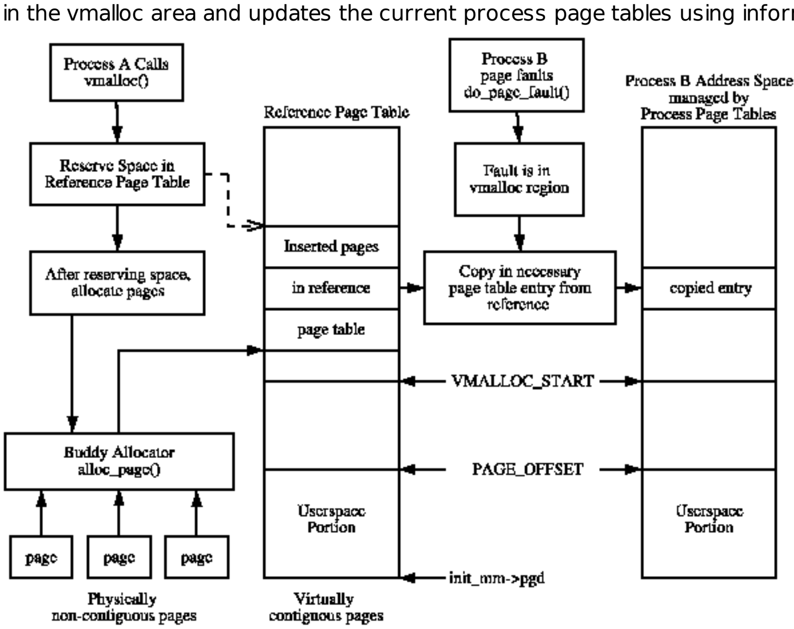




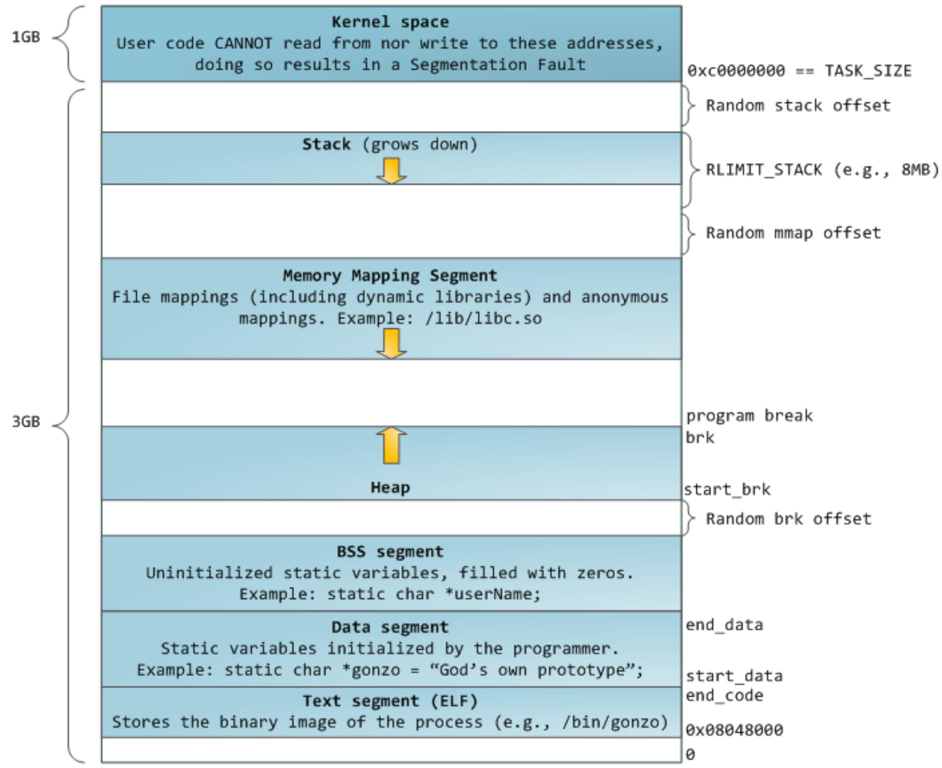
Why vmalloc can not be used in interrupt routine?

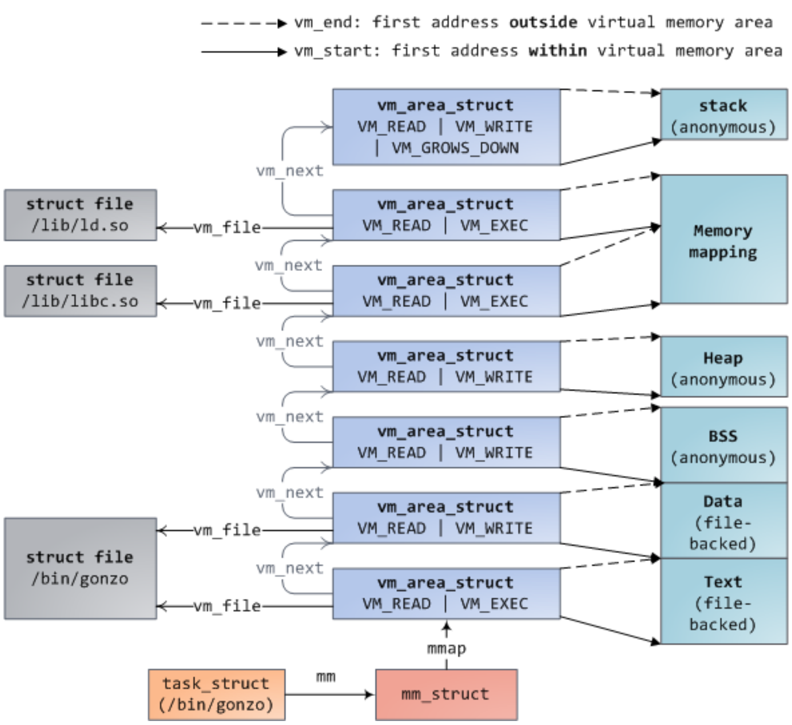
1.vmalloc will call kmalloc(GFP_KERNEL) to allocate page table, which will cause process to sleep.

2.vmalloc will cause page fault
The page table updated by `vmalloc()` is not the current process but the reference page table stored at `init_mm→pgd`. This means that a process accessing the vmalloc area will cause a page fault exception as its page tables are not pointing to the correct area. There is a special case in the page fault handling code which knows that the fault occured in the vmalloc area and updates the current process page tables using information from the master page table.

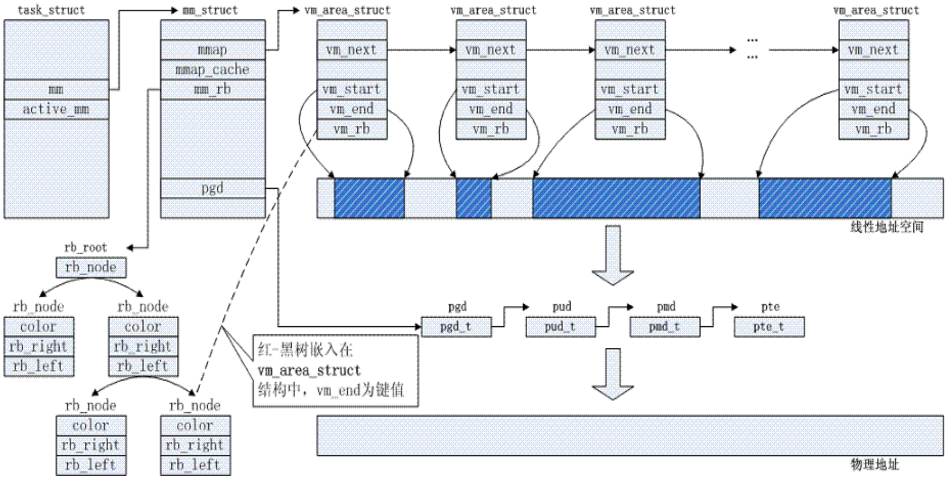


7. Process Address space





vm_area_struct



http://duartes.org/gustavo/blog/post/how-the-kernel-manages-your-memory/

How to check the vm_area_struct

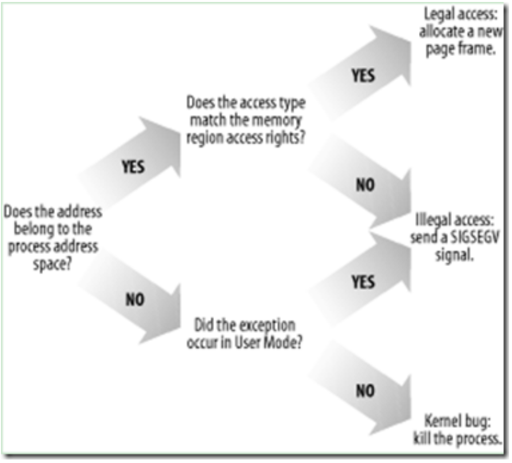
```
[root@pek2-office-18th-10-117-174-219 work]# pmap -x 11749
11749:   ./loop
Address           Kbytes     RSS   Dirty Mode  Mapping
0000000000400000       4       4       0 r-x-- loop
0000000000600000       4       4       4 r---- loop
0000000000601000       4       4       4 rw--- loop
0000003d91200000     132     128       0 r-x-- ld-2.17.so
0000003d91420000       4       4       4 r---- ld-2.17.so
0000003d91421000       4       4       4 rw--- ld-2.17.so
0000003d91422000       4       4       4 rw---   [ anon ]
0000003d91600000    1752     640       0 r-x-- libc-2.17.so
0000003d917b6000    2048       0       0 ----- libc-2.17.so
0000003d919b6000      16      16       8 r---- libc-2.17.so
0000003d919ba000       8       8       8 rw--- libc-2.17.so
0000003d919bc000      20      12      12 rw---   [ anon ]
00007f19d5d6d000      12      12      12 rw---   [ anon ]
00007f19d5d90000       4       4       4 rw---   [ anon ]
00007fff41214000     132      12      12 rw---   [ stack ]
00007fff4124f000       8       0       0 r----   [ anon ]
00007fff41251000       8       4       0 r-x--   [ anon ]
ffffffffff600000       4       0       0 r-x--   [ anon ]
crash> vm
PID: 2596   TASK: ffff88007b0e1540  CPU: 0   COMMAND: "crash"
       MM               PGD          RSS    TOTAL_VM
ffff88007c4f2d00  ffff88007d4e5000  828332k  972200k
      VMA           START       END     FLAGS FILE
ffff88007d4e3530     400000     a1e000 8001875 /usr/bin/crash
ffff88007d4e35f8     c1d000     c3f000 8101873 /usr/bin/crash
ffff88007d4e36c0     c3f000     dd5000 100073
ffff88007d4e3788     e3e000     e5e000 8101873 /usr/bin/crash
ffff880078935ae8    17f3000   298c8000 100073
ffff88007c36d3e0 302ae00000 302ae02000 8000075 /lib64/libdl-2.12.so
ffff88007c36d318 302ae02000 302b002000 8000070 /lib64/libdl-2.12.so
ffff88007c36d4a8 302b002000 302b003000 8100071 /lib64/libdl-2.12.so
```

mprotect

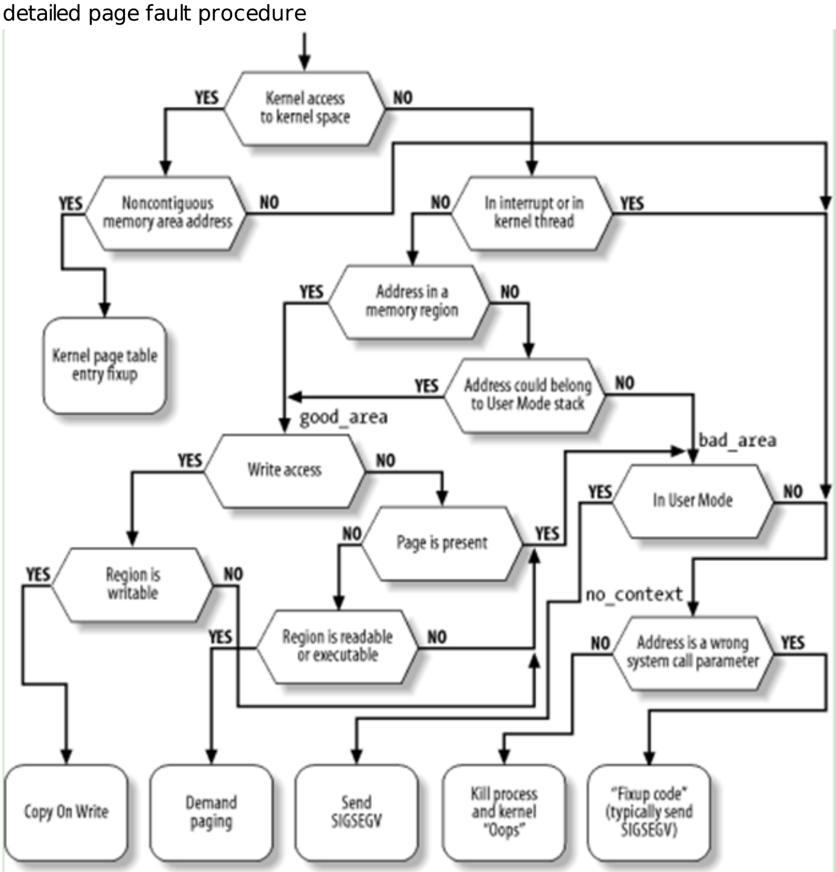
NAME
       mprotect - set protection on a region of memory
SYNOPSIS
       #include <sys/mman.h>
       int mprotect(void *addr, size_t len, int prot);

Page Fault

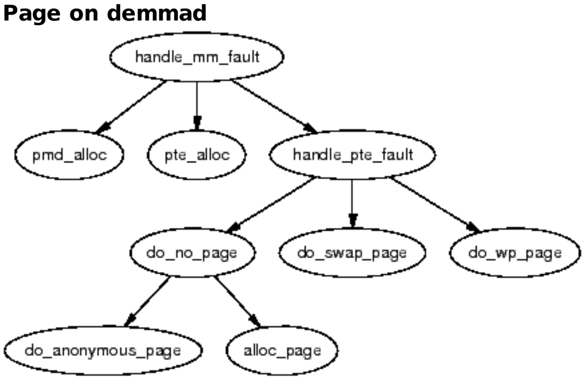
general procedure



detailed page fault procedure



Page on demmad



new and malloc()

how about this code?

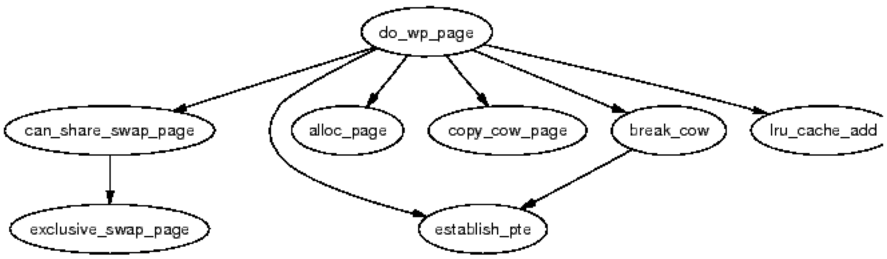
int *p = malloc(1024*1024*sizeof(int));

for(i = 0; i < 4096*1024;i++){

```
    printf("%d", p[i]);
}
```

In this case, the system-wide `empty_zero_page`, which is just a page of zeros, is mapped for the PTE and the PTE is write protected.

COW



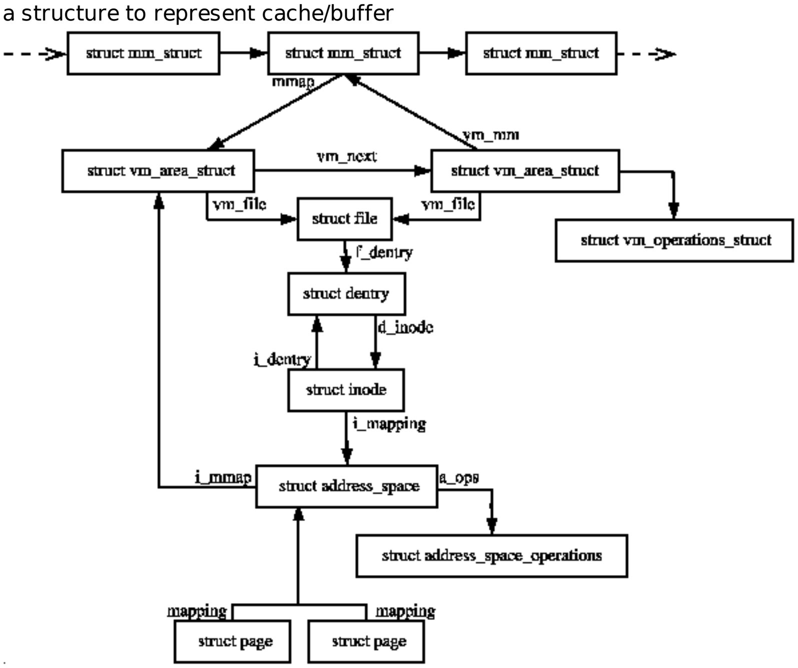
Linux recognises a COW page because even though the PTE is write protected, the controlling VMA shows the region is writable.

8. Memory Cache/Buffer

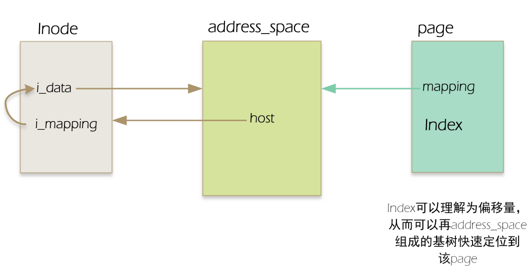
address_space

what's a address_space?

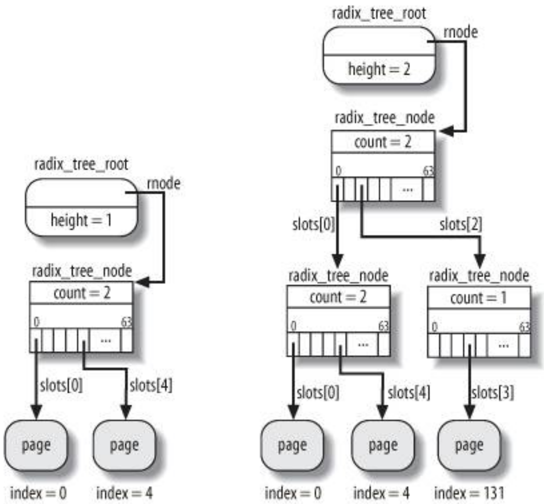
a structure to represent cache/buffer



.



radix tree



(a) radix tree of height 1 (b) radix tree of height 2

radix tree tagging
mark a rb-node and all its parent recursively until root if the one of its page if dirty or writeback

share memory
tmpfs based
mapped the tmpfs file to process's address space

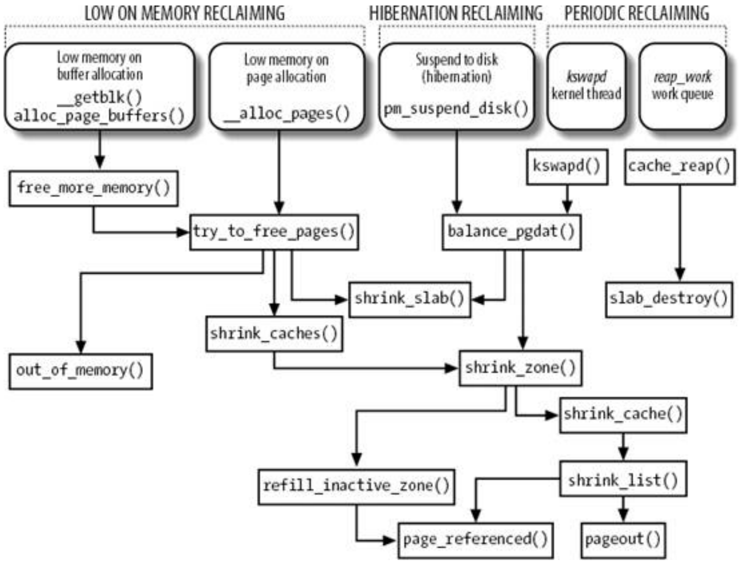
https://www.kernel.org/doc/gorman/html/understand/understand015.html

9. PFRA
which page to be reclaimed?

Type of pages	Description	Reclaim action
Unreclaimable	Free pages (included in buddy system lists) Reserved pages (with `PG_reserved` flag set) Pages dynamically allocated by the kernel Pages in the Kernel Mode stacks of the processes Temporarily locked pages (with `PG_locked` flag set) Memory locked pages (in memory regions with `VM_LOCKED` flag set)	(No reclaiming allowed or needed)
Swappable	Anonymous pages in User Mode address spaces Mapped pages of *tmpfs* filesystem (e.g., pages of IPC shared memory)	Save the page contents in a swap area
Syncable	Mapped pages in User Mode address spaces Pages included in the page cache and containing data of disk files Block device buffer pages Pages of some disk caches (e.g., the inode cache)	Synchronize the page with its image on disk, if necessary
Discardable	Unused pages included in memory caches (e.g., slab allocator caches) Unused pages of the dentry cache	Nothing to be done

Algorithm??
second chance and lru combined
active referenced

How to reclaim?



10. SWAP

to be continued...