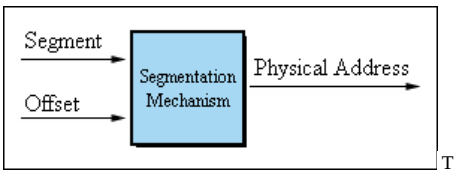
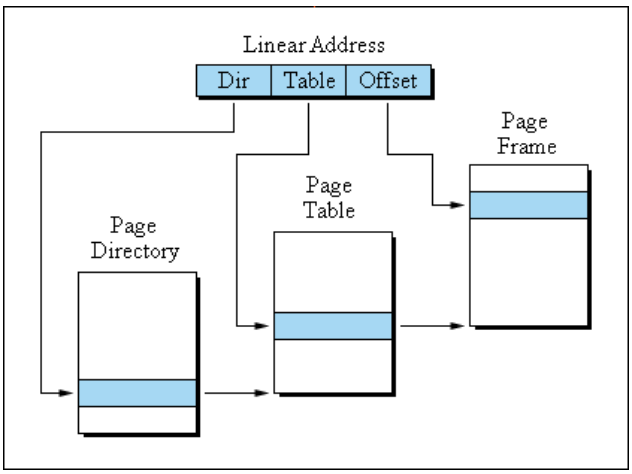
Real Mode VS Protected mode

Real Mode



T

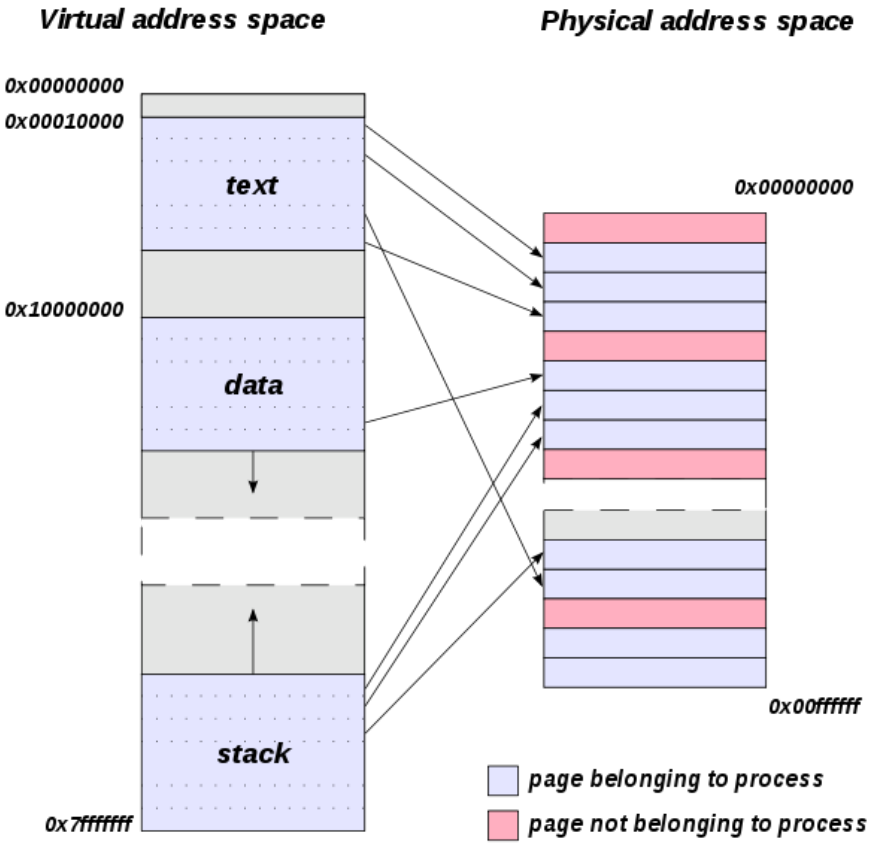
Protected mode



How to enter protected mode

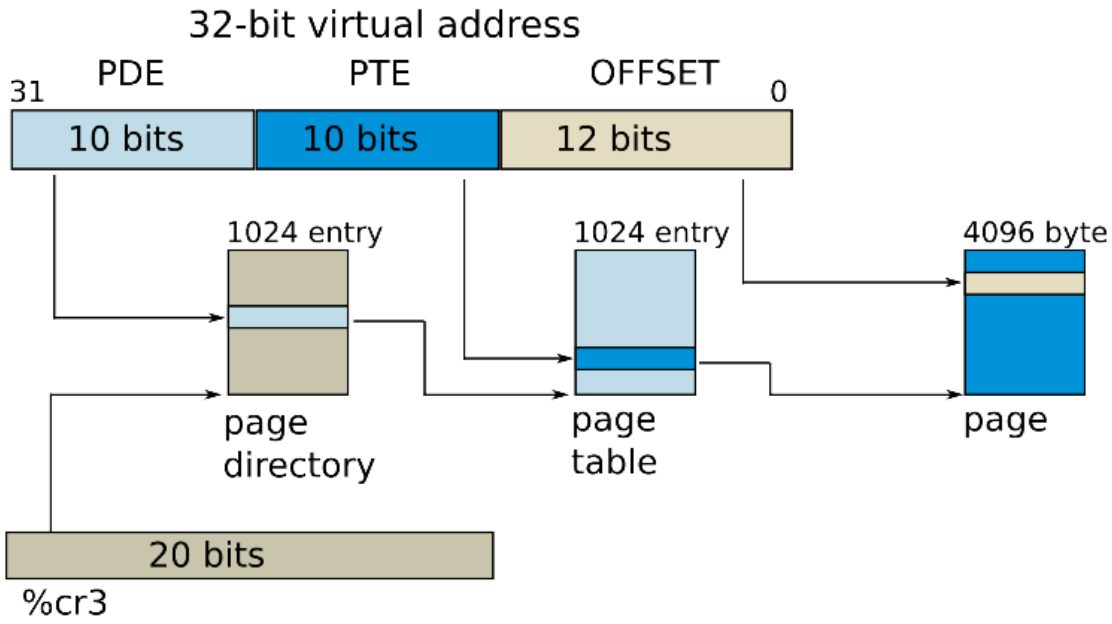
```
; set PE bit
mov eax, cr0
or eax, 1
mov cr0, eax ;
far jump (cs = selector of code segment)
jmp cs:@pm

@pm: ; Now we are in PM.
```

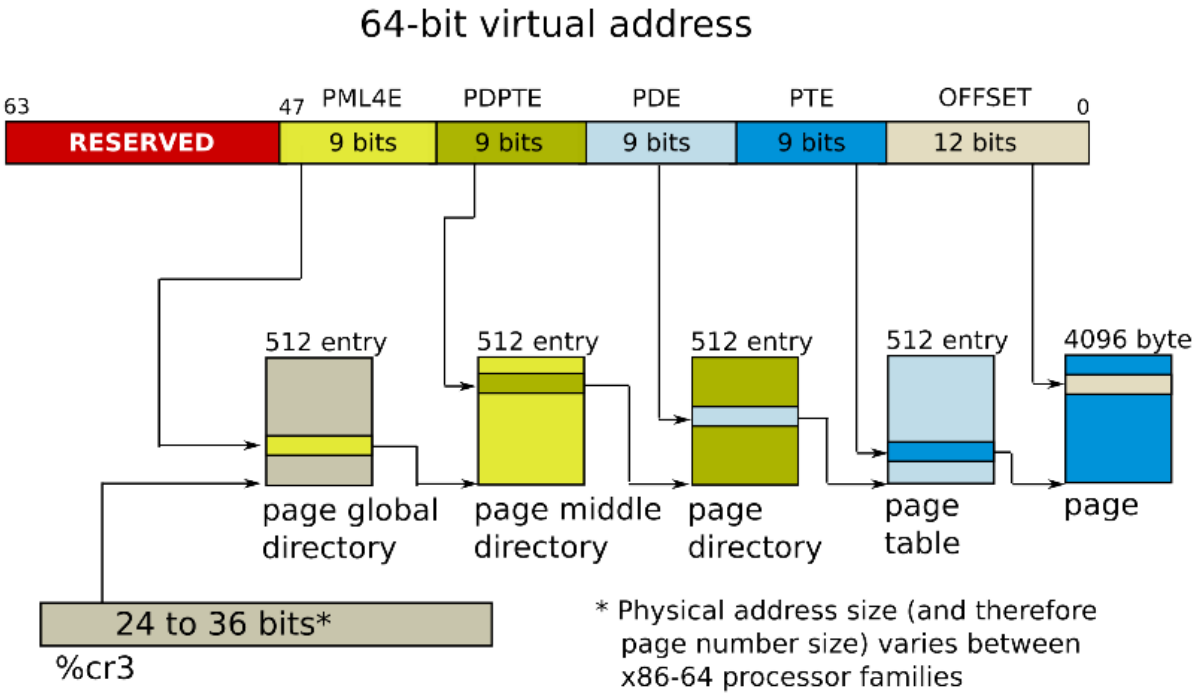
what's page table used for?



Common method of using paging to create a virtual address space

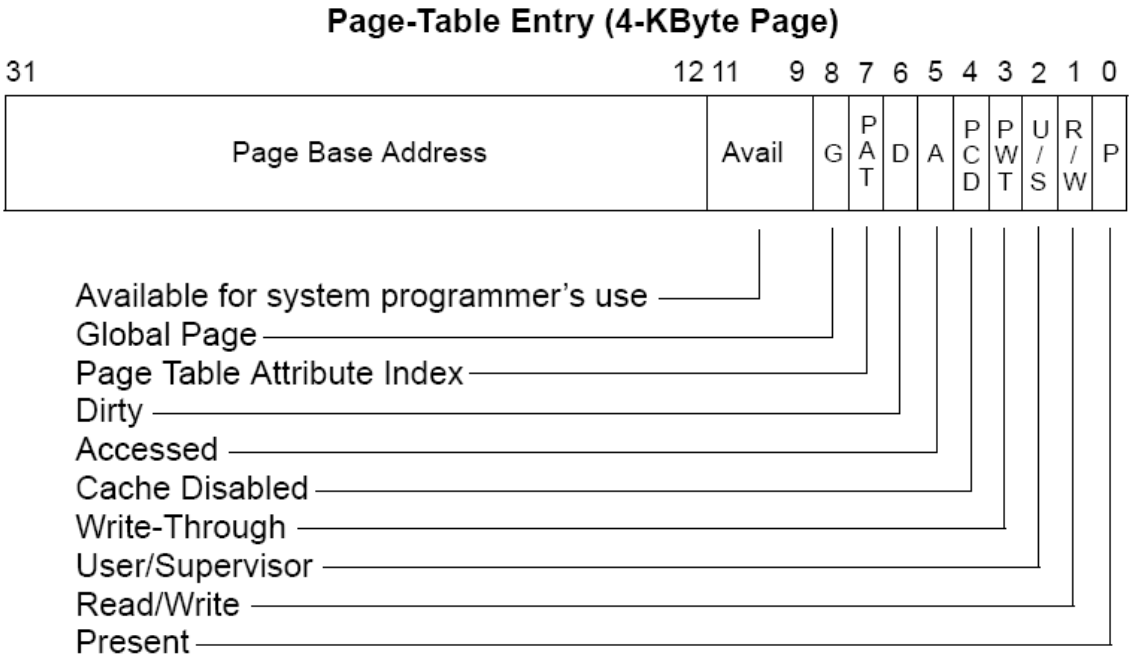
Page Table in 32 and 64 bit architecture



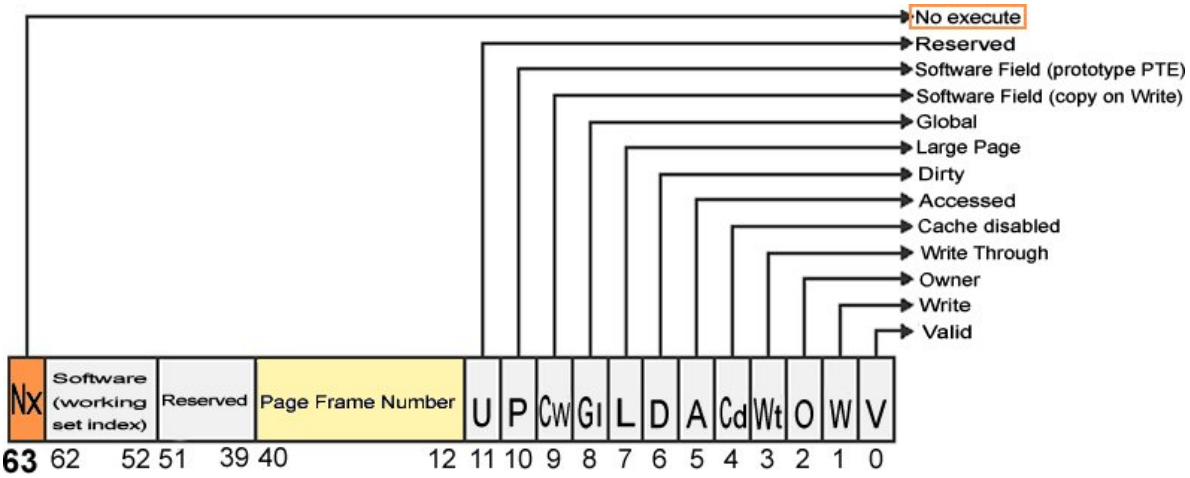


PTE

32bit



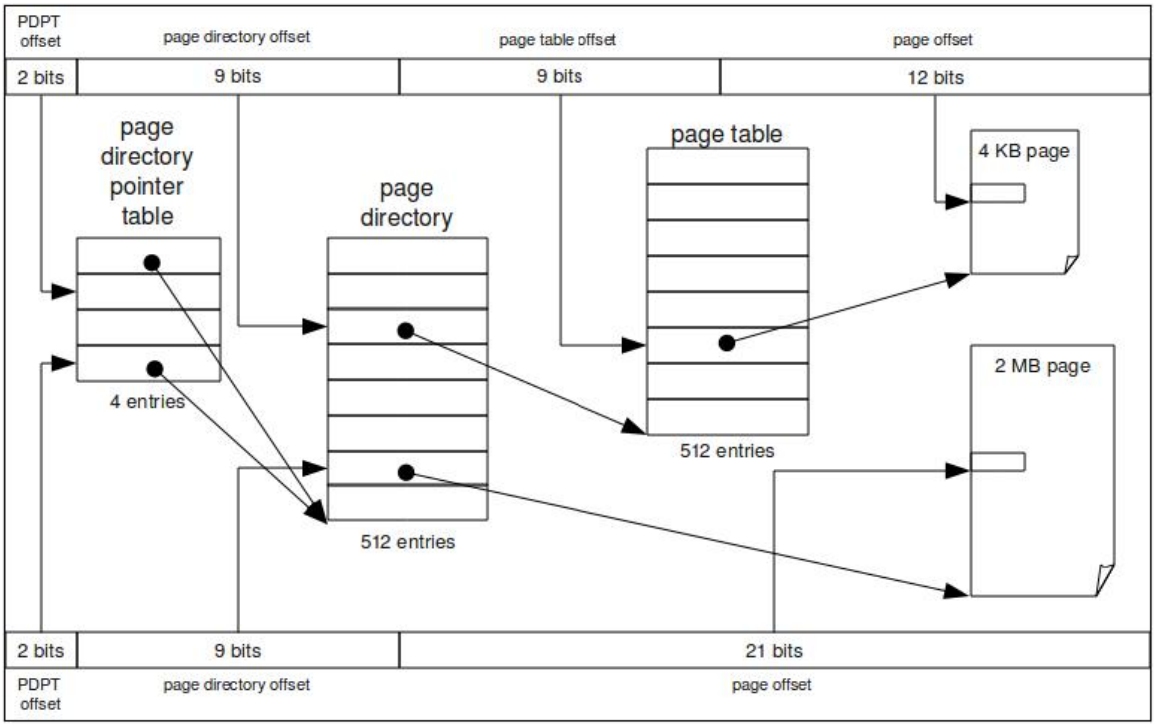
64 bit



Bit	Function
_PAGE_PRESENT	Page is resident in memory and not swapped out
_PAGE_PROTNONE	Page is resident but not accessable
_PAGE_RW	Set if the page may be written to
_PAGE_USER	Set if the page is accessible from user space
_PAGE_DIRTY	Set if the page is written to
_PAGE_ACCESSED	Set if the page is accessed

Huge page vs Transparent Huge Page

why huge pages????



1.Huge page
http://linuxgazette.net/155/krishnakumar.html

2.Transparent Huge Page
http://lwn.net/Articles/423584/
THP tries to make huge pages "just happen" in situations where they would be useful.
The current patch only works with anonymous pages; the work to integrate huge pages with the page cache has not yet been done.
The main kernel address space is mapped with huge pages, reducing TLB pressure from kernel code.

How to dump kernel page tables?

1.add CONFIG_X86_PTDUMP=y to .config
2.recompile kernel
3.mount debug fs
mount -t debugfs debugfs /debug
4.less /debug/kernel_page_tables

```
---[ User Space ]---
0x0000000000000000-0xffff800000000000    16777088T                     pgd
---[ Kernel Space ]---
0xffff800000000000-0xffff880000000000        8T                     pgd
---[ Low Kernel Mapping ]---
0xffff880000000000-0xffff880000099000      612K     RW           GLB NX pte
0xffff880000099000-0xffff88000009a000        4K     ro           GLB NX pte
0xffff88000009a000-0xffff88000009b000        4K     ro           GLB x  pte
0xffff88000009b000-0xffff880000200000     1428K     RW           GLB NX pte
0xffff880000200000-0xffff880001000000       14M     RW       PSE GLB NX pmd
0xffff880001000000-0xffff880001600000        6M     ro       PSE GLB NX pmd
```

Crash command : vtop ptov

```
crash> vtop 302ae00000    <---linear address
VIRTUAL     PHYSICAL
302ae00000  775fa000        <---physical address


   PML: 7c914000 => 7c90c067
   PUD: 7c90c600 => 7ca0d067
   PMD: 7ca0dab8 => 7ca0f067
   PTE: 7ca0f000 => 775fa005  <----005 belongs to page offset, it's used for flag.
  PAGE: 775fa000  <---page address,


  PTE     PHYSICAL  FLAGS
775fa005  775fa000  (PRESENT|USER)


     VMA           START       END     FLAGS FILE
ffff88007c9ca188 302ae00000 302ae02000 8000075 /lib64/libdl-2.12.so


     PAGE       PHYSICAL      MAPPING       INDEX CNT FLAGS
ffffea0001a1ceb0 775fa000 ffff88007b68c5d8        0 71 2000000002006c

crash> vtop ffff88007c778e90
VIRTUAL           PHYSICAL
ffff88007c778e90  7c778e90


PML4 DIRECTORY: ffffffff81a85000
PAGE DIRECTORY: 1a86063
   PUD: 1a86008 => 8067
```

PMD: 8f18 => 800000007c6001e3
PAGE: 7c600000 (2MB)

PTE	PHYSICAL	FLAGS
800000007c6001e3	7c600000	(PRESENT|RW|ACCESSED|DIRTY|PSE|GLOBAL|NX)

PAGE	PHYSICAL	MAPPING	INDEX	CNT	FLAGS
ffffea0001b3a240	7c778000	0	14	1	20000000000080

Idea

1. force a process to swap out some page, then get the pte from the address, find the swap slot position, read it out???
2. page table takes lots of memory because of share memory?
3. alloc pte may cause a process to sleep!!!

```
static int alloc_pte_page(pmd_t *pmd)
{

 pte_t *pte = (pte_t *)get_zeroed_page(GFP_KERNEL | __GFP_NOTRACK);

 if (!pte)

 return -1;

 set_pmd(pmd, __pmd(__pa(pte) | _KERNPG_TABLE));

 return 0;
}
```

copy from user?