ITP 30002 Operating System

Paging: Smaller Tables

OSTEP Chapter 20

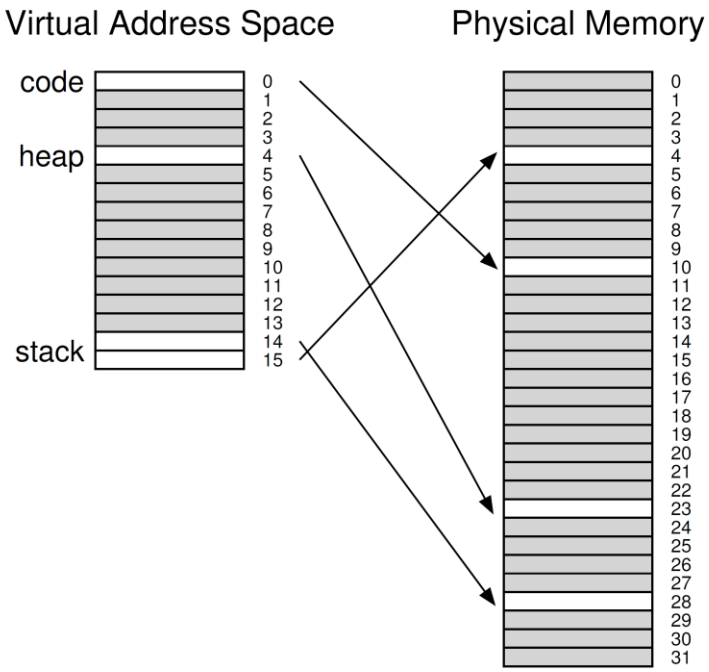
Shin Hong

Make Page Table Smaller

- Array-based page tables take too much memory resource even though most of page entries are invalid
  - e.g., for a 32-bit address space with 4 KB pages, a per-process page table takes 4 MB

- Approaches
  1. use bigger pages
  2. per-segment page tables
  3. multi-level page tables
  4. inverted page table

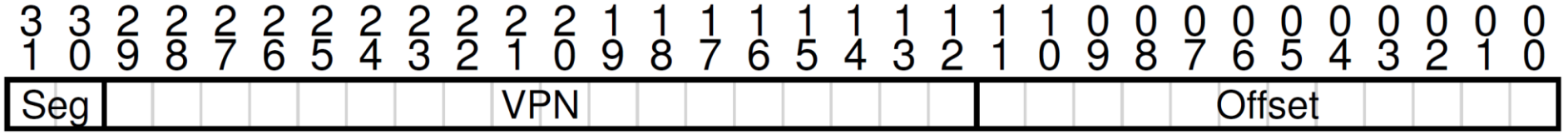
Per-segment Page Table



PFN	valid	prot	present	dirty
10	1	r-x	1	0
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
23	1	rw-	1	1
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
28	1	rw-	1	1
4	1	rw-	1	1

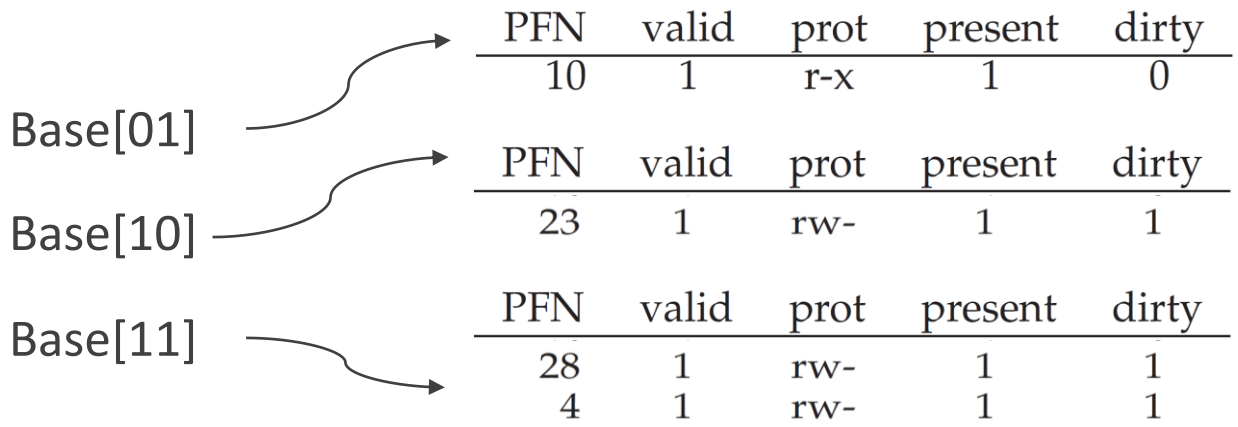
- Based on the observation that only first few pages of each segment is used in many programs
- Have three page tables for three segments, and allocate a variable-length memory region for holding a page table
  - the base register points to the beginning of a page table, and the bounds register represents the number of allocated pages in the segment

Example



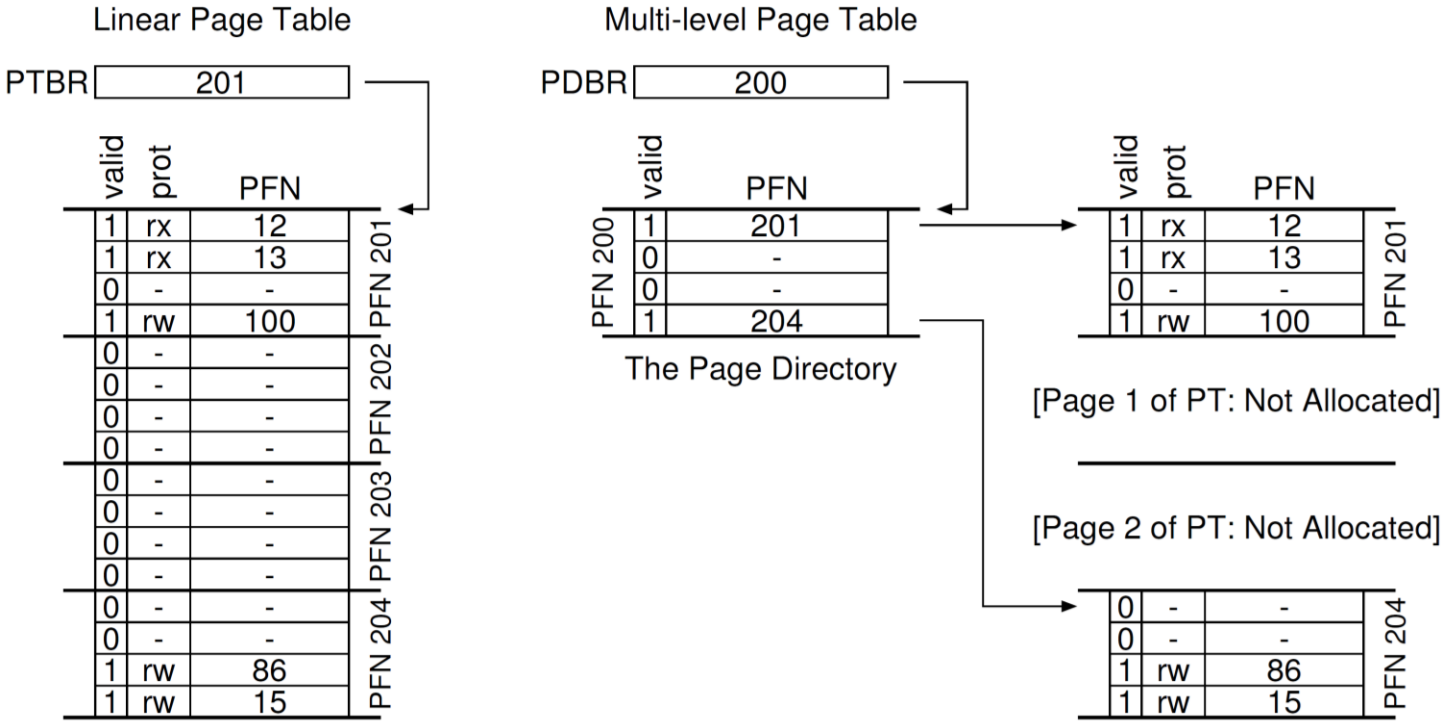
- use the two bits to represent a segment identifier
  - 01 for code, 10 for the heap, 11 for the stack
- access a page table segment at a TLB miss

```
SN            = (VirtualAddress & SEG_MASK) >> SN_SHIFT
VPN           = (VirtualAddress & VPN_MASK) >> VPN_SHIFT
AddressOfPTE = Base[SN] + (VPN * sizeof(PTE))
```



Multi-level Page Tables

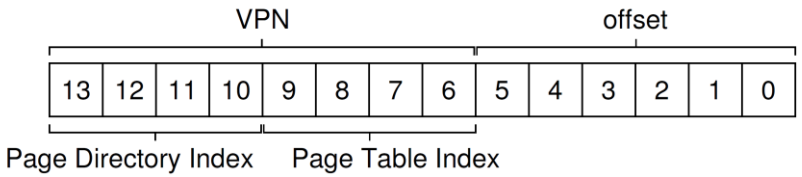
- Divide a page table into page-size units
- Allocate a page to each page table piece, but do not allocate a page if the corresponding unit has no valid entry
- Have a **page directory** as an index of the allocated pages for a page table
- E.g.



Example

- Address size of 16 KB (2^{14}) with 64-byte pages
  - 256 VPNs
  - a page table takes 1 KB (16 pages) if a PTE takes 4 bytes
- Suppose that VPNs only 0, 1, 4, 5, 254 and 255 are used, and the rest are unused

0000 0000	code
0000 0001	code
0000 0010	(free)
0000 0011	(free)
0000 0100	heap
0000 0101	heap
0000 0110	(free)
0000 0111	(free)
.....	... all free ...
1111 1100	(free)
1111 1101	(free)
1111 1110	stack
1111 1111	stack



PDEAddr = PageDirBase + PDIndex * sizeof(PDE)

PTEAddr = (PDEAddr->PFN << SHIFT)
+ PTIndex * sizeof(PTE)

Page Directory

PFN	valid?
100	1
—	0
—	0
—	0
—	0
—	0
—	0
—	0
—	0
—	0
—	0
—	0
—	0
—	0
—	0
—	0
101	1

Page of PT (@PFN:100)

PFN	valid	prot
10	1	r-x
23	1	r-x
—	0	—
—	0	—
80	1	rw-
59	1	rw-
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—

Page of PT (@PFN:101)

PFN	valid	prot
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
—	0	—
55	1	rw-
45	1	rw-

Two-level Page Table Control Flow


```
1  VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2  (Success, TlbEntry) = TLB_Lookup(VPN)
3  if (Success == True)   // TLB Hit
4    if (CanAccess(TlbEntry.ProtectBits) == True)
5      Offset   = VirtualAddress & OFFSET_MASK
6      PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7      Register = AccessMemory(PhysAddr)
8    else
9      RaiseException(PROTECTION_FAULT)
10 else                   // TLB Miss
11   // first, get page directory entry
12   PDIndex = (VPN & PD_MASK) >> PD_SHIFT
13   PDEAddr = PDBR + (PDIndex * sizeof(PDE))
14   PDE     = AccessMemory(PDEAddr)
15   if (PDE.Valid == False)
16     RaiseException(SEGMENTATION_FAULT)
17   else
18     // PDE is valid: now fetch PTE from page table
19     PTIndex = (VPN & PT_MASK) >> PT_SHIFT
20     PTEAddr = (PDE.PFN << SHIFT) + (PTIndex * sizeof(PTE))
21     PTE     = AccessMemory(PTEAddr)
22     if (PTE.Valid == False)
23       RaiseException(SEGMENTATION_FAULT)
24     else if (CanAccess(PTE.ProtectBits) == False)
25       RaiseException(PROTECTION_FAULT)
26     else
27       TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
28       RetryInstruction()
```

Inverted Page Table

- Keep a single page table that has an entry for each frame
  - each frame is mapped to a VPN of a process
  - only one page table would be enough for a single system

- Searching the entry for a VPN of a process consumes much more time than array-based page tables
  - linear search, hashing, etc.