

Paging: Introduction

Operating System (OS)
안인규

Concept of Paging

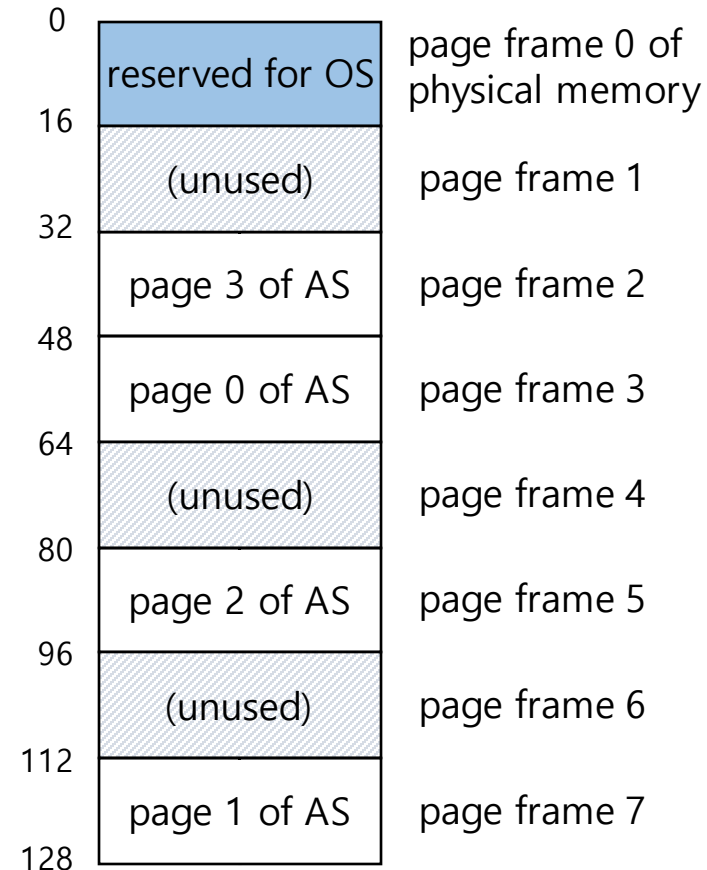
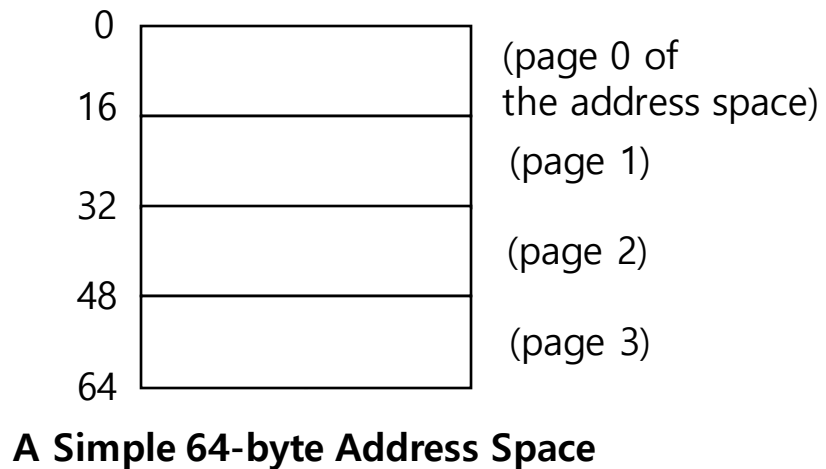
- Paging **splits up** address space into **fixed-sided** unit called a **page**.
 - Segmentation: variable size of logical segments(code, stack, heap, etc.)
- With paging, **physical memory** is also **split** into some number of pages called a **page frame**
- **Page table** per process is needed **to translate** the virtual address to physical address.

Advantages Of Paging

- **Flexibility:** Supporting the abstraction of address space effectively
 - Don't need assumption how heap and stack grow and are used.
- **Simplicity:** ease of free-space management
 - The page in address space and the page frame are the same size.
 - Easy to allocate and keep a free list

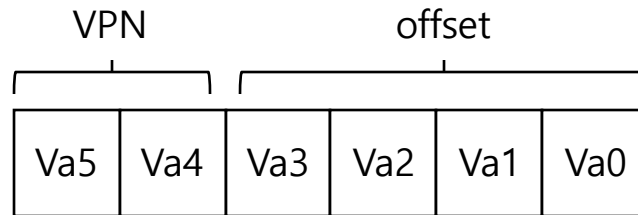
Example: A Simple Paging

- 128-byte physical memory with 16 bytes page frames
- 64-byte address space with 16 bytes pages

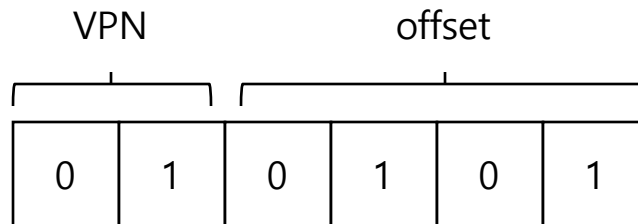


Address Translation

- Two components in the virtual address
 - VPN: virtual page number
 - Offset: offset within the page

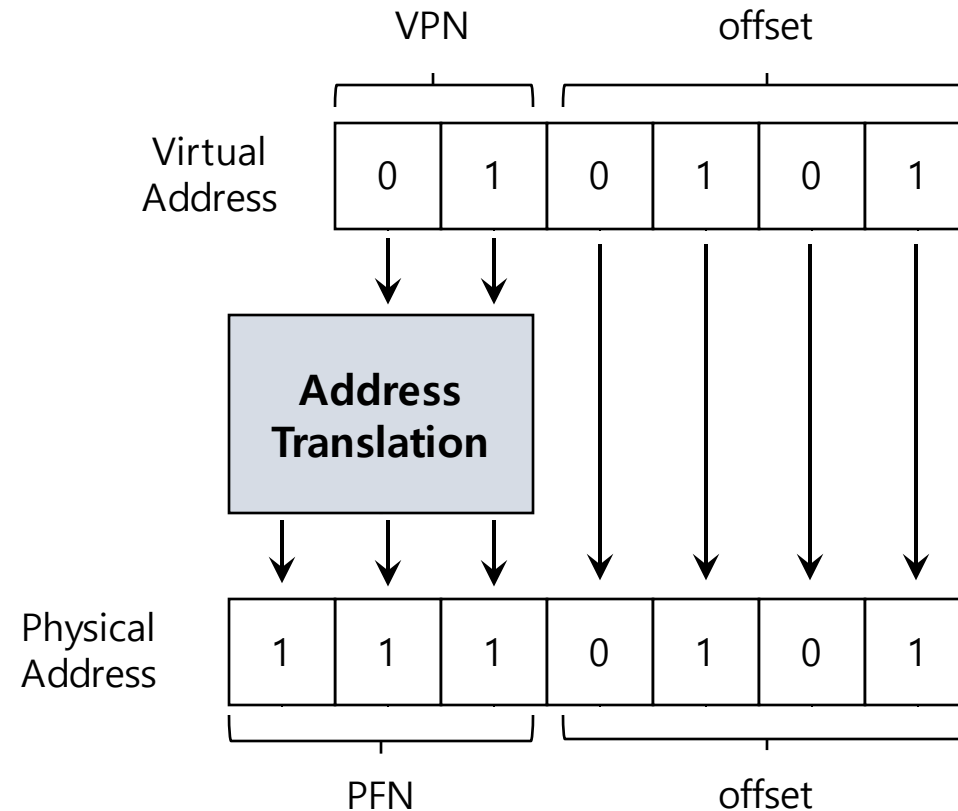


- Example: virtual address 21 in 64-byte address space



Example: Address Translation

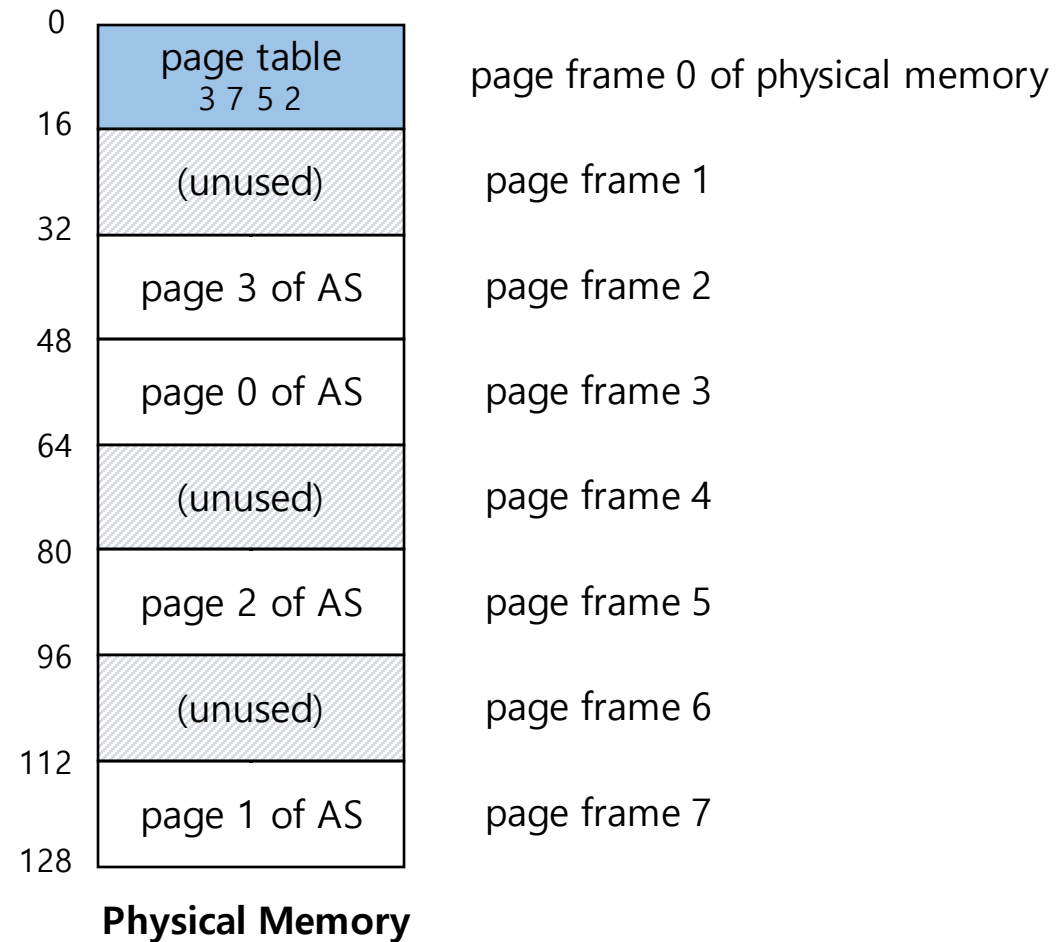
- The virtual address 21 in 64-byte address space



Where Are Page Tables Stored?

- Page tables can get awfully large.
 - 32-bit address space with 4-KB pages, 20 bits for VPN
 - Page offset for 4 Kbyte page: 20 bit
 - $4MB = 2^{20}$ entries * 4 Bytes per page table entry
- Page tables for each process are stored in memory

Example: Page Table in Kernel Physical Memory



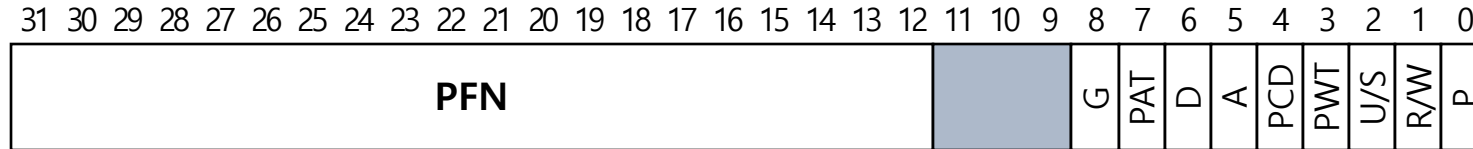
What Is In The Page Table?

- The page table is a **data structure** that is used to map the virtual address to physical address.
 - Simplest form: a linear page table, an array
- The OS **indexes** the array by VPN, and looks up the page-table entry.

Common Flags Of Page Table Entry

- **PFN (Page Frame Number):** 이 가상 페이지가 매핑된 물리 프레임 번호
- **Valid bit:** 이 페이지 매핑이 유효한지 (0이면 접근 시 오류 발생)
- **R/W (Read/Write):** 쓰기 허용 여부 (0이면 읽기 전용)
- **U/S (User/Supervisor):** 사용자/커널 권한 여부 (0이면 커널만 접근 가능)
- **Present bit:** 이 페이지가 현재 메모리에 존재하는지 (0이면 디스크에 있음)
- **Accessed bit (A):** 이 페이지가 최근에 접근됐는지
- **Dirty bit (D):** 수정(쓰기)된 적이 있는지

Example: x86 Page Table Entry



An x86 Page Table Entry(PTE)

- P: present
- R/W: read/write bit
- U/S: supervisor
- A: accessed bit
- D: dirty bit
- PFN: the page frame number

Paging: Too Slow

- To find a location of the desired PTE, the **starting location** of the page table is **needed**.
- For every memory reference, paging requires the OS to perform one **extra memory reference**.

Accessing Memory With Paging

```
1 // Extract the VPN from the virtual address
2 VPN = (VirtualAddress & VPN_MASK) >> SHIFT
3
4 // Form the address of the page-table entry (PTE)
5 PTEAddr = PTBR + (VPN * sizeof(PTE))
6
7 // Fetch the PTE
8 PTE = AccessMemory(PTEAddr)
9
```

Accessing Memory With Paging

```
10 // Check if process can access the page
11 if (PTE.Valid == False)
12     RaiseException(SEGMENTATION_FAULT)
13 else if (CanAccess(PTE.ProtectBits) == False)
14     RaiseException(PROTECTION_FAULT)
15 else
16     // Access is OK: form physical address and fetch it
17     offset = VirtualAddress & OFFSET_MASK
18     PhysAddr = (PTE.PFN << PFN_SHIFT) | offset
19     Register = AccessMemory(PhysAddr)
```

A Memory Trace

```
int array[1000];  
...  
for (i = 0; i < 1000; i++)  
    array[i] = 0;
```

```
prompt> gcc -o array array.c -Wall -o  
prompt> ./array
```

Memory access

```
0x1024 movl $0x0, (%edi,%eax,4) // [edi+eax*4]= 0  
0x1028 incl %eax // eax++  
0x102c cmpl $0x03e8,%eax // 0000 0011 1110 10002 = 100010  
0x1030 jne 0x1024
```

A Virtual(And Physical) Memory Trace

