page table을 아예 Disk에 저장하자.

메모리의 크기보다 큰 프로세스를 실행하려면 how?

Beyond Physical Memory: Mechanisms

Prof. Yongtae Kim

Computer Science and Engineering
Kyungpook National University

Swap Space

- **Realistically, an address space doesn't fit into physical memory**
  - OS needs a place to stash away portions of address spaces that currently aren't in great demand ➔ additional level in the memory hierarchy (e.g. disk)

숨기는 장소

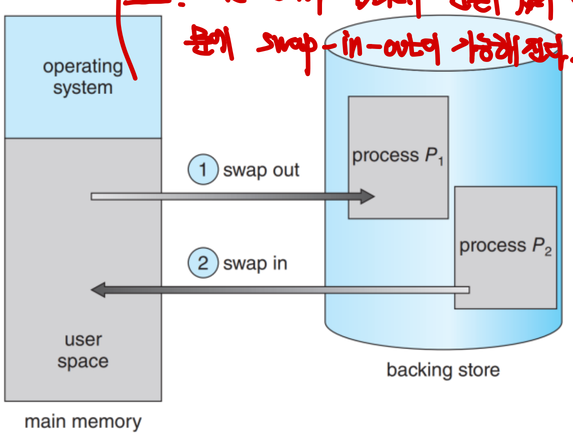
잠깐 던더 놓은것

- **For moving pages back and forth, OS reserves some space, refer to as "swap space" on the disk**

→스왑 공간.

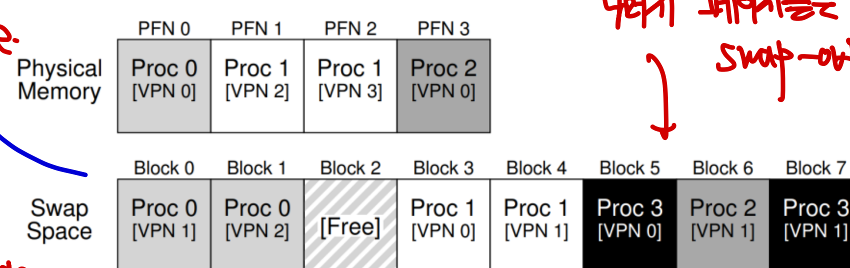
  - OS can read from and write to the space in page-sized units
  - OS needs to remember the disk address of a given page →OS는 page들의 디스크에서의 주소를 다 저장해야 함.
- **The swap space is not the only on-disk location for swapping traffics (e.g. code pages)**

(코드 영역은 어차피 Disk의 원본이 있기 때문에 swap-in-out이 가능해진다.) 4page.



proc3은 다 swap-out : 실행중이 X

나머지 페이지들은 swap-out



8page swap space

→ 실제 메모리보다 더 많은 공간이 존재하는 것처럼 가정할 수 있다.

The process 0~3 are sharing memory (some pages are in swap space)

The Present Bit~~w~~

→ 주소 변환을 하려고 할 때 page가 메모리에 있는지, swap공간에 있는지도 봐야 한다.

→ TLB s/s에서는 어떻게 처리할까.

- **Overview of the memory reference**
  - The running process generates a virtual address
  - The hardware extracts VPN, checks TLB, and obtains physical address if hit
  - Otherwise, the hardware locates the page table in memory and looks up PTE
  - If page is valid and present in memory, the hardware obtains physical address

    ↳ present bit에 저장

    → 메모리에 존재하지 않는다면 page fault → page fault handler

    ↳ page를 swap공간에서 찾도록

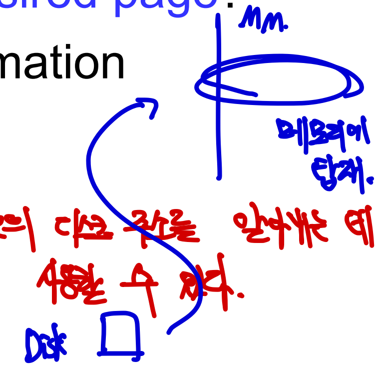
- **To allow pages to be swapped to disk, more machinery is needed**
  - When the hardware looks in the PTE, it may find that the page is not present in physical memory and this information is known as the present bit in the PTE

Present Bit	Meaning
0	The page is not in memory but rather on disk somewhere
1	The page is present in physical memory

- **The act of accessing a page that is not in physical memory is commonly referred to as a page fault**
  - Upon a page fault, OS is invoked to service the page fault (page fault handler)

Page Fault

- **The page faults are handled by software (OS) for the systems regardless of hardware- or software-managed TLBs**
  - Note that TLB miss handling is done in differently according to TLB type
- **If a page is not present and has been swapped to disk, OS will need to swap the page into memory to service the page fault**
  - 어디서 찾아오는 건가.
  - Then, the question is how will OS know where to find the desired page?
  - Usually, the page table is a natural place to store such information



- **When the OS receives a page fault for a page:**

PTE bit로 page의 디스크 주소를 알아내는 데 사용할 수 있다.

1) Looking in the PTE to find a disk address for the page
2) Issuing the request to disk to fetch the page into memory
3) During disk I/O to swap the page in ➔ the process to be in the blocked state
4) Updating the page table to mark the page as present in memory
   present bit 0→1
5) Retrying the instruction ➔ This may result in a TLB miss
6) Updating the TLB with the new entry
7) Restarting the instruction and this will be a TLB hit ➔ Finally, memory access!

Page Fault Control Flow

- **What hardware does during the address translation:**

HW base 에즈
exception을 발생

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success, TlbEntry) = TLB_Lookup(VPN)
if (Success == True)    // TLB Hit
    if (CanAccess(TlbEntry.ProtectBits) == True)
        Offset   = VirtualAddress & OFFSET_MASK
        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
        Register = AccessMemory(PhysAddr)
    else
        RaiseException(PROTECTION_FAULT)
else                    // TLB Miss
    PTEAddr = PTBR + (VPN * sizeof(PTE))
    PTE = AccessMemory(PTEAddr)
    if (PTE.Valid == False)
        RaiseException(SEGMENTATION_FAULT)
    else
        if (CanAccess(PTE.ProtectBits) == False)
            RaiseException(PROTECTION_FAULT)
        else if (PTE.Present == True)
            // assuming hardware-managed TLB
            TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
            RetryInstruction()
        else if (PTE.Present == False)
            RaiseException(PAGE_FAULT)
```

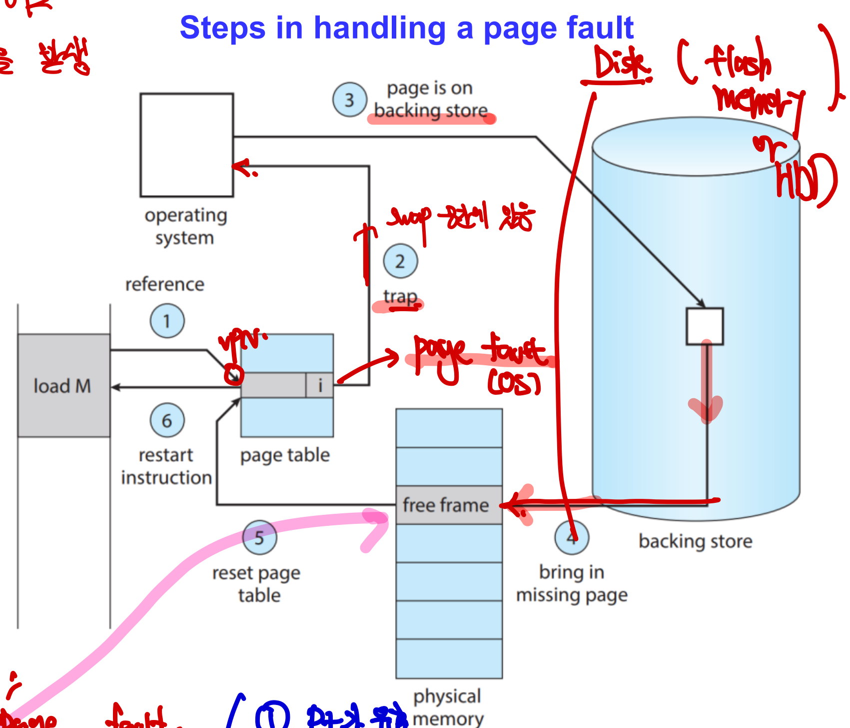
TLB miss

valid & present

valid & !present

page fault.

① PTE가 有 ② present가 false 여야 함



- **What OS does upon a page fault:**

```
PFN = FindFreePhysicalPage()
if (PFN == -1)              // no free page found
    PFN = EvictPage()       // run replacement algorithm
DiskRead(PTE.DiskAddr, PFN) // sleep (waiting for I/O)
PTE.present = True          // update page table with present
PTE.PFN     = PFN           // bit and translation (PFN)
RetryInstruction()          // retry instruction
```

PTE의 disk 정보.

OS to find a physical frame for the soon-be-faulted-in page to reside within and run replacement algorithm if failed

When Page Replacement Occur?

- **When memory is full of pages, OS needs to swap old page(s) in memory out to disk to make room for new page(s) in disk**
  - The process of picking a page to kick out, or replace is known as the page-replacement policy

- **When replacements really occur?**
  - To keep a small amount of memory free, OS usually has high watermark (HW) and low watermark (LW) to decide when to start evicting pages from memory
  - If (# of free pages < LW), a background thread that is responsible for freeing memory runs to evicts pages and goes to sleep if (# of free pages > HW)
  - This background thread is called swap daemon or page daemon

얘들이 LW보다 free page가 적으면 swap; HW보다 많아질 때까지

- **The performance optimizations become possible by performing a number of replacements at once** → page cluster → Disk I/O 비용 최적화.
  - Many systems will cluster or group a number of pages and write them out at once to the swap partition, thus increasing the efficiency of the disk